2002

Landmark navigation of the titan 4WD outdoor mobile robot using narrow beam CTFM sonar

Danny Ratner
University of Wollongong

Recommended Citation
NOTE

This online version of the thesis may have different page formatting and pagination from the paper copy held in the University of Wollongong Library.

UNIVERSITY OF WOLLONGONG

COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.
8 Recognition of Isolated landmarks

The term recognition in this research is used in conjunction with isolated landmarks for the technical process involved in finding the best match between the measured signatures collected in the scanning (during angular or linear sweep of isolated landmark) to the reference signature of the landmark in the recognition map.

8.1 Introduction

The purpose of this chapter is to explain the methodology used to develop robust landmark recognition for isolated landmarks. Titan's navigation strategies are inspired from the experience of flying light aircraft during cross-country navigation. Light aircraft will navigate – in a good weather condition (VFR) – a set of legs between isolated landmarks (prominent landscape, river bend, major junction etc.) or legs of continuous landmarks (river, road, coast line). In bad weather, pilots will fly blind legs using Instruments Flying Rules (IFR).

When the pilot of light aircraft conducts a VFR leg between isolated landmarks, the pilot uses the landmarks to calibrate his navigation plan. At each landmark, the recognition of the landmark provides the pilot with information about the accumulated localisation error and enables him to take the appropriate steps to correct the navigation. In our research, Titan uses an angular or linear sweep to scan the landmark and assess its localisation error.

Bats may use very similar flying and sensing modes for navigation and hunting. When flying in open spaces they use their biological odometry and clock to estimate distance. Flying in a cave or in plantation they fly along continuous landmarks (cave walls or line of trees). To locate a tiny prey they scan the target by moving their ears while transmitting the FM chirps.

Research conducted by Harper and McKerrow [1999 a] showed that using 19 features extracted from the CTFM Acoustic Density Profile of small plants, in an indoor laboratory set-up, could be used to classify plants into groups. In the Titan project, we extended the research on plant classification by harnessing the CTFM sensing technology
to the recognition of landmarks for outdoor navigation. Isolated landmarks may be plants (bushes, trees trunk) or any prominent object (pole, banch, equipment) available along the planned path. The size of the potential landmark should be within reasonable proportion with the scanned field (angular section of 50 degree or radial section of 1250 mm). The challenge we were facing included noisy outdoor background (the landmark is always cluttered by grass and other plants and objects), changing environment with respect to time compared to the controlled indoor structured environment, motion of the robot relative to environment and limited CPU resources due to multi-tasking of the robot control programs during navigation.

8.2 Features extraction from scanned data

The Titan robot is equipped with a recognition map which consists of a set of reference CTFM signatures (one reference signature for each isolated landmark). Each reference signature was taken at the nominal 90° pan angel and a tilt angel of 0 or 5° in most experiments, (depends on the tested environment). To conduct recognition of an isolated landmark, the robot performs, at the start of each leg, an angular or a linear sweep of the expected landmark, and stores a collection of two features from each of the 51 successive measured CTFM signatures. Finding the correct best match of the measured scanned data to the recorded data in the recognition map is the essence of the recognition.

To achieve an efficient recognition process the search focuses on two defined features which are extracted from the scanned data in real time. The first feature is defined as the “Relative Intensity” and it reflects the level of energy in the signal. The second feature is defined as “Pattern Similarity” and it reflects the level of correlation between the measured CTFM signature and the reference CTFM signature.

The experiments conducted showed us that extraction of the pattern similarity feature is not enough to determine the correct matching. The relative intensity feature
teaches us to distinguish between strong reflection areas in the sweep (probably the landmark) and weak reflections areas in the sweep (probably the background). But the peak of the relative intensity curve does not always coincide with the location of the reference signature. Sometimes a small surface of a highly specular texture can generate a very strong reflection. Regarding the pattern similarity feature, experiments taught us that diffusive textures like outdoor grass and indoor carpet generate random values which can be confusing. The combination of the two features is successfully applied via signal processing which includes windowing and thresholding and helps to achieve the correct recognition.

8.3 Thresholding considerations

The thresholding of the relative intensity feature creates a window in the sweep which can be used to remove non relevant areas of the similarity feature from the recognition search space. The experiments proved that the relative intensity level can be used to distinguish potential landmark sections of the sweep from the background. We found that there is a dramatic change in the relative intensity between any typical landmark and the common background noise. The investigation of the signature stability conducted in Chapter 6 shows that the relative intensity plots have considerable fluctuations with the time, but this has no significant effect on the basic ratio of target to noise. On the other hand, the plot of the pattern similarity is much more stable and will provide good distinction when smartly searching for suspected targets.

The results of this investigation leads to the idea of establishing a reasonable thresholding level for the relative intensity. The region of the relative intensity above this threshold creates a window on the pattern similarity feature. This window helps to reduce the susceptibility of the pattern matching to misleading values from random reflections coming from diffusive backgrounds. The value of the threshold was determined by experiments in the typical environment and in essence reflects the ratio of target to noise encountered in the environment during navigation. A rule of thumb of 10% usually yielded good results. We have to bear in mind that 10% still enables robust distinction even if
there are intensity fluctuations. If the landmark is a corridor wall the intensity from the wall could be thousands of time bigger than the reflection from the carpet on the floor. In outdoor environments the ratios are much more modest but still can be around 50 or 100 times. The level of the background can be easily seen in the figures in chapter 6.

8.4 Angular sweep at the start of the leg

The angular sweep is performed at the start of an isolated landmark leg (the navigation map includes an index telling the robot which kind of a leg is expected and the appropriate recognition strategy). The sweep session is conducted while the robot is motionless, thus enabling it to channel all the CPU resources to support the effort of recognition (compare it to a driver standing at a junction concentrating on scanning the area while trying to identify the correct position). The experience gathered in this mode is illustrated in 3 experiments presented and analysed using Matlab programs for post processing. The first experiment (Figures 8.1 to 8.6 generated by AngularSweepSonarProcess1.m) describes angular scanning of a wooden pole stuck in the grass. The second experiment (Figures 8.7 to 8.10 generated by AngularSweepSonarProcess2.m) describes angular scanning of metal pole with a round base on the carpet in the lab.

The third experiment (Figures 8.11 to 8.13 AngularSweepSonarProcess3.m) describes angular scanning of an old LabMate robot equipped with a ring of ultrasonic and infrared sensors on the carpet in the lab. In all of these experiments the robot was not moved between the recording of the reference signature and the following angular sweep, thus conducting recognition at a known position. The zero range difference found in the recognition process just confirm this set up.

In additional experiments (including the experiments with landmark temporarily cluttered with objects and linear scanning) - localisation errors were introduced and the consistency of the recognition was confirmed. It is interesting to note that the first two experiments were conducted with simple round poles which are axis symmetric targets. A
The target in the third experiment has no axis-symmetry and therefore the information from the pattern similarity plot yields exactly the 90° pan point where the reference was sensed. We have to bear in mind that from practical reasons, the position of the target (especially referring to pole type target) with respect to the robot is around 90° by best manual judgement. This is the possible reason for the deviation of the recognition point from 90°. Other factors for the deviation may be that a pole stuck in the grass and somewhat inclined. The only way to assure that the pole target is placed exactly at 90° is to hang the pole to ensure totally vertical alignment and conduct a scan and move the pole so the maximum intensity will coincide with a pan angle of 90°. In practice, this is not needed for a converging correction of the navigation.

In the first experiment, a 22.5mm diameter pole was stuck in the grass. Figure 8.1 shows the relative intensity, the window produced by thresholding it at 10% and the pattern similarity. Notice how the window reduces the search space. The pole was recognised to be at scan position 31. In Figure 8.2, the range difference at this position is zero, while away from the pole the system measures the range to the grass.

For completeness, we include a full set of figures for this experiment. Figure 8.3 shows the reference signature, and Figure 8.4 shows all 51 measured signatures. These are plotted as linear sonargrams in Figure 8.5, and a log sonargram in figure 8.6. Because the signatures in Figure 8.4 are normalised, they do as show the large amplitude variation that is observable in the sonargrams.

However, the sequence of signatures in Figure 8.4 becomes a single spike around the recognition point because the signal to noise ratio of the target is very high. In contrast, the signatures at the ends of this sequence show a high level for the echoes from the grass.
Figure 8.1 Angular sweep of an isolated landmark – experiment 1. Relative intensity feature normalised (solid line) and pattern similarity feature (max-auto-correlation = 1) (dashed line) from a wooden pole stuck in the grass. Max. relative intensity is scan 29 while best windowed (10% threshold) pattern similarity is scan 31.
Figure 8.2 Angular sweep of an isolated landmark – experiment 1. Range difference measurement along the angular sweep of a wooden pole stuck in the grass. Note that the range difference is flat around the recognition point of scan No. 31. Range difference is extracted by comparing auto/cross-correlation of reference/measured signatures.
Figure 8.3 Angular sweep of isolated landmark – experiment 1. Reference CTFM signature of a wooden pole stuck in the grass. Signature is recorded after averaging 15 successive readings at 90° pan and 5° tilt angle.
Figure 8.4 Angular sweep of isolated landmark – experiment 1. Angular sweep of a wooden pole stuck in the grass consists of 51 normalised measured signatures starting at 65° and finishing at 115° at 1° pan angle steps. X axis is the 128 elements equivalent to 0-7500 Hz. Best match by windowed pattern similarity is signature 31 (starting from top-left of the figure which is tilted at 90 degree) with xcorr = 0.9998.
Figure 8.5 Angular sweep of isolated landmark – experiment 1. Linear sonogram from an angular sweep of a wooden pole stuck in the grass.
Figure 8.6 Angular sweep of isolated landmark – experiment 1. Log sonagram from an angular sweep of a wooden pole stuck in the grass.

In the second experiment, a 64mm diameter metal pole is supported by a base on a carpet indoors. The greater diameter of the pole results in a wider window in Figure 8.7 (as compared to figure 8.1). Also, the pattern similarity graph is much flatter and higher when the background is carpet rather than grass. The reason for this is that the carpet
produces less background signal when the sensor is not pointing directly at the pole. Also, there are no variations in range because the reflectors from the pole dominates the reflector from the carpet at all angles (Figure 8.8)

Figure 8.7 Angular sweep of isolated landmark – experiment 2. Relative intensity feature normalised (solid line) and pattern similarity feature (max-auto-correlation = 1) (dashed line) from a red metal pole on a carpet. Max. relative intensity is scan 28 while best
windowed pattern similarity is scan 31. Window on pattern similarity plot is set by a 10% threshold on relative intensity plot.

Figure 8.8 Angular sweep of isolated landmark – experiment 2. Range difference measurement along the angular sweep of a red metal pole on a carpet. Note that the range difference is always flat around the recognition point (scan No. 31 in this case), but because the metal pole is such a strong target compared to the carpet - the range is flat along all the sweep. Range difference is extracted by comparing auto/cross-correlation of reference/measured signatures. The smooth metal pole is a very dominant target even on the far field of the carpet, so the pattern of the measured signature retains its shape along the sweep.
Figure 8.9 Angular sweep of isolated landmark - experiment 2. Angular sweep of a red metal pole with a round base on a carpet consists of 51 normalised measured signatures starting at 65° and finishing at 115° at 1° pan angle steps. X axis is the 128 elements equivalent to 0-7500Hz. Best match by windowed pattern similarity is signature 31 (starting from top-left) with xcorr = 0.99491047859200. To plot - run AngularSweepSonarProcess2.m. The metal pole is very effective target and is dominant even in the far field of the sweep.
Figure 8.10 Angular sweep of an isolated landmark – experiment 2. Sonagram from an angular sweep of a red metal pole with a round base on a carpet on a log scale.
In the third experiment, a complex geometry object (semi symmetrical Labmate robot) was insonified. The relative intensity graph (Figure 8.11) is not as sharp as that for the poles. The pattern similarity graph is much more variable. The range to the target (Figure 8.12) changes significantly throughout the sweep due to the complex geometry of the target.

The target is considered to be at scan 27 (Figure 8.11), the peak of the pattern similarity inside the window. Note, this doesn’t align with the peak of the relative intensity, which is at scan 29. This example shows the importance of using the relative intensity to window the pattern similarity. As shown in the sonargram (Figure 8.13) the acoustic density profile of the target is much more complex.
Figure 8.11 Angular sweep of an isolated landmark – experiment 3. Relative intensity feature normalised (solid line) and pattern similarity feature (max-auto-correlation = 1) (dashed line) from a LabMate robot uncluttered by red metal pole. Max. relative intensity is scan 29 while best windowed pattern similarity is scan 27. Notice the high level of the pattern similarity from background area that would confuse the recognition if window had not been set to remove effects of background areas.
Figure 8.12 Angular sweep of an isolated landmark – experiment 3. Range difference measurement along the angular sweep from a LabMate robot on a carpet uncluttered by red metal pole. Note that the range difference is flat around the recognition point of scan No. 27. Range difference is extracted by comparing auto/cross-correlation of reference/measured signatures. The LabMate robot was not moved during the experiment which include recording reference signature and conducting angular sweep to verify recognition.
Figure 8.13 Angular sweep of an isolated landmark – experiment 3. Linear sonargram of the Labmate robot used as a target. The assembly of 51 acoustic density profiles (CTFM signatures) creates a complex image reflecting the complex shape of the LabMate geometry.

8.5 Linear sweep session at the start of the leg

The angular sweep session is performed without robot motion. The idea behind the linear sweep is to use the robot’s motion to conduct the scanning of the area of the expected landmark and save the need to a stop and conduct a panning sequence. The
combination of motion and recognition dictates special consideration on the issues of process execution time management. The continuous acquisition of the audio signal used in angular sweep was changed to distance-triggered single acquisition (with timed-delayed splitted sequence of filling and reading) as described in Chapter 5.

Motion control of a very low velocity on rough outdoor surfaces is problematic due to the poor resolution of the odometry sensors. Variable velocity affects the synchronisation of the distance dependent triggering. If the velocity of the robot fluctuates, the results are multiple signatures and even absent signatures. Another aspect of a linear sweep is that it adds an inherent error to the leg distance, since the robot stops only after extending the nominal leg distance by half a sweep length. At the end of the sweep, the best match is found and the sonar head will pan towards the recognition point. Localisation is done with the same triangulation algorithm used with the angular sweep (Chapter 2.13 – Figures 2.23 and 2.24).

The following 3 experiments illustrate the feasibility of linear sweep. The 3 successive experiments describe linear scanning of a wooden pole stuck in the grass. The robot was moved to a different localisation error before the start of each of the linear sweep resulting in a different recognition matching of the reference signature.

In the first experiment (Figures 8.14-8.16) the pole is at the left end of the graphs in Figure 8.14. That is near the start of the linear sweep. In this situation, the range to the pole (Figure 8.15) is at its lowest when the sensor is pointing at the pole, because it is an isolated landmark. The linear sonargram (Figure 8.16) shows how dominant the pole is.

In the second experiment (Figure 8.17), the pole is toward the middle of the linear sweep. In the third experiment (Figure 8.18), the pole is toward the end of the linear sweep. The fact that the results are the same for the three experiments indicates that quality of the approach.

The post processing for the first experiment was done by LinearSweepSonarProcess1.m), for the second experiment by LinearSweepSonarProcess2.m) and for the third experiment by LinearSweepSonarProcess3.m).
Figure 8.14 Linear sweep experiment number 1. Linear sweep of relative intensity feature normalised (solid line) and pattern similarity (dashed line) feature (max-auto-correlation = 1) from a wooden pole stuck in the grass. Maximum relative intensity is located at scan 7 (0.15 meter) while maximum pattern similarity occurs at scan 10 (0.225 meter).
Figure 8.15 Linear sweep experiment number 1. Range difference measurements from a wooden pole stuck in the grass. Note that the range difference is flat around the recognition point at scan No. 10. Range difference is extracted by comparing auto/cross-correlation of reference/measured signatures. The robot was given initial random localisation (linear and angular) error.
Figure 8.16 Linear sweep experiment number 1. Linear sonargram showing the location of the pole located at the start of the sweep.
Figure 8.17 Linear sweep experiment number 2. Linear sweep of relative intensity (solid line) and pattern similarity (dashed line) features from a wooden pole stuck in the grass. Maximum relative intensity is located at scan 28 (0.675 meter) while maximum pattern similarity occurs at scan 27 (0.65 meter).
8.6 Recognition in a cluttered environment

The angular sweep experiment of the LabMate target in the lab (angular sweep experiment number 3 - Figures 8.11 to 8.13) was extended for an angular sweep of...
LabMate robot in a cluttered environment. The purpose was to illustrate the recognition capability of a complex shape landmark (LabMate robot equipped with sensor ring) which is temporarily cluttered by another object. For this illustration, a red metal pole with a round base was added to the expected area of the landmark.

First a reference signature from the cluttered landmark (including the pole) is used (Figures 8.19) to confirm that the normal recognition process works. The reference signature was recorded with the sonar pointing toward the LabMate not the pole. After recording the signature the robot was moved to give it some lateral localisation error. The metal pole has a very strong reflection but despite this fact a correct recognition is established.

In Figure 8.19, we see that two windows are formed from the relative intensity. However, the pattern similarity is much higher in the window representing the LabMate (left) resulting in correct recognition.

Second, a reference signature was recorded the LabMate with the clutter removed. In this experiment, our aim was to recognise the cluttered landmark with respect to the planned uncluttered reference signature. We found from many experiments that reflections from surfaces with diffusive texture (lab’s carpet in this case) could produce very misleading values of pattern similarity. In this example, the peak of the pattern similarity (Figure 8.20) yields an erroneous recognition result but when used in combination with the relative intensity feature the correct position is extracted.

Figure 8.20 shows the results from this new search. The 10% threshold of the relative intensity graph creates two windows – the pole area (right) and the LabMate area (left). The pattern similarity has higher value with respect to the left window which is associated with the LabMate target than with the pole. It is interesting to note that without the windowing the recognition point will be scan 12 (To the left of the LabMate and the pole) instead of the correct value of scan 28. This experiment shows that the recognition is robust enough to ignore the distraction from powerful reflectors that may clutter the area.
Figure 8.21 compares the reference signatures of the LabMate used in the two experiments, showing the influence of the pole on left signature. The sonargram in Figure 8.22 shows how much the pole (left) dominates the LabMate (right and behind).

![Graph showing relative intensity and pattern similarity](image)

**Figure 8.19** Angular sweep experiment number 4. Relative intensity feature (solid line) and pattern similarity feature (max-auto-correlation = 1) (dashed line) from a LabMate robot cluttered by red metal pole with a round base. Reference signature taken with cluttered landmark. Max. relative intensity is scan 50 while best windowed pattern similarity is scan 28.
Figure 8.20 Reproducing the sweep of the relative intensity (solid line) and pattern similarity (dashed line) features by cross-corelating the uncluttered reference signature (recorded without pole) with the 51 measured signatures (with pole in the far field). Maximum relative intensity is scan number 50 while best windowed (above 10% threshold intensity) pattern similarity is scan number 28. Without using windowing the recognition point will be scan number 12 which is erroneous result.
Figure 8.21 The two reference signatures used in this experiment. The signature on the left is with the LabMate cluttered with the pole while that on the right is a LabMate without the presence of the pole. Note that the metal pole positioned in the far field has very strong echo which still capable to influence the reference signature of the LabMate recorded at mid field (90 degrees).
Figure 8.22 Linear sonagram of the LabMate target (represented by the lower shape) cluttered by a 64 mm pole (represented by a very high shape). The experiments showed the capability of Titan to recognize the recorded LabMate landmark and not being distracted by the temporary introduction of the pole to the field of audition.
9 Recognition of continuous landmarks

The term recognition in conjunction with continuous landmarks relates to the detection of the edge of a path. It is a continuous process combined with motion and thus provides a continuous flow of data regarding the location of the edge. Continuous landmarks do not have a reference signature in the recognition map. The flow of data provides feedback for the motion controller instead of using bearing from the compass in the feedback control.

9.1 Introduction

As mentioned earlier, Titan’s navigation strategies were inspired by the experience gained in navigation of light aircraft. During good visibility condition, the pilot will maintain Visual Flight Rules (VFR) using recognition of landmarks to calibrate his navigation. VFR navigation combines flying between isolated landmarks with following of continuous landmarks. The decision to use set of isolated landmarks or to use a continuous landmark depends upon the availability of landmarks along the desired route. Continuous landmarks should be prominent objects or landscape compared to the surrounding area. Natural continuous landmarks that are used frequently include coast line landscapes and rivers. Structured objects like roads or power lines are common continuous landscapes in a featureless scenery (like desert or wide plans).

In this chapter, we discuss navigating Titan along a continuous landmark using CTFM ultrasonic sensing to correct the trajectory instead of the human visual servoing done by the pilot. When flying along a continuous landmark, the pilot continuously detects the edge of the landmark and corrects any accumulated deviation from the desired relative position. The recognition challenge in this mode is to keep in touch with the edge. This process is done concurrently with motion and therefore the allocation of the pilot attention is mainly to keep the aircraft flying and regularly glance at the edge of the landmark. The pilot does not need to monitor the compass to maintain magnetic bearing. Time is monitored only occasionally to estimate the aircraft position within the leg. In the case of flying, the pilot is exposed to optical flow of the continuous landmark. In the case of a bat flying in a cave, it may use acoustic flow.
Titan also regularly senses the edge using motion control. The robot senses the path edge every 0.25 meters. The distance triggered sensing (with a single split acquisition and reading) enables the allocation of sufficient CPU resources to motion control. The robot uses extracted features to track the edge. The compass is not used in this mode. Travelled distance is monitored to know the distance needed to finish the leg. The underlying assumption in this research is that a smooth path is flanked by a rough verge.

9.2 Exploring sensing strategies for edge detection

We explored a few sensing strategies in order to achieve robust and efficient edge detection in indoor and outdoor environments. Firstly, we investigated the performance of edge detection when employing a similar approach to the recognition of isolated landmark. The first approach (tested using LabVIEW implementation SonarCapture1.vi combined with SonarAnalyzer1.vi) compared the measured CTFM signature at each distance interval to a typical reference signature of the edge (recognition map contained one reference signature per continuous landmark leg). The second approach was to record (implemented by SonarCapture2.vi combined with SonarAnalyzer2.vi) a sequence of reference signatures (the recognition map contains specific reference signature for each distance interval) and compare each measured signature with its supposed counterpart.

A few difficulties were encountered with these methods. The edge of a walking path does not always maintain the same signature pattern. A sequence of many reference signatures is very cumbersome and incurs a CPU expensive operation when interacting with such a recognition map. There are also synchronisation problems when so many reference signatures are involved and robot accumulates some distance errors. In addition, experiments conducted along smooth walls (corridors or road kurbs) exposed us to the problem of multi-path reflections. In such case, the similarity of measured signature and reference signature can become very poor. In the process of recognising isolated landmarks we used the information from the relative intensity to window relevant search sections in the pattern similarity. When searching for a path edge, the search involves only one measured signature compared to 51 measured signatures. The third approach was devised to solve these difficulties.
Figure 9.1 Screen dump taken during the navigation shows the signature of the grass. The flat area at the start of the signature represents the concrete path.

The chosen approach (implemented by SonarCapture4.vi combined with SonarAnalyzer4.vi) is not using any reference signature. Instead, each measured CTFM signature (which is taken every 0.25 meters) is analysed to find the first peak above certain threshold (implemented by FindFirstPeak3.vi). We found that the signature is composed of a near-zero flat section (the concrete path way) followed by a relative step of strong signature (the grass or garden - Figure 9.1). This transition is common to many kinds of edges, but each with a different power threshold. In addition, we found that the absolute intensity (not the relative intensity, as in the case of scanning isolated landmarks) indicates whether the robot is looking at a smooth path or at a mixed area of path and grass.

9.3 Features extraction from scanned data

The main goal of the distance-triggered sensing is to keep in touch with the path edge. Detection of a path edge is achieved by exploiting the different spectral response of the surfaces across the edge. The concrete path (for example) behaves like a mirror (specular texture) by reflecting the energy away, but the grass behaves like a diffusive
surface since each leaf is a small reflector. To realise robust recognition, Titan extracts two features from the CTFM sensing of the path edge.

The first feature is the "corner" in the CTFM signature. The corner is a sharp step in the acoustic density profile which occurs at the edge of the path, where the geometric shape of the surface changes. This corner is created by the spatial filtering of the flat concrete path which reflects the energy away and effectively acts like a low band filter. The signature evolves only from the edge into the grass area. The area of the path is cut out. The corner is found by searching for the first peak above a threshold.

The second feature is the intensity of the signal. A certain level of intensity is maintained while sensing the edge. If intensity drops below certain level, the meaning is that the robot is loosing the edge because it has moved too deeply into the concrete path and the program initiates a predefined range error which turns the robot towards the edge.

9.4 **Multi-path reflections when following a corridor wall**

The wall of a corridor is a flat and smooth surface and therefore it behaves like a mirror target to ultrasonic waves. It is a simple continuous landmark, but there are some difficulties caused by its spectral features compared to following a complex type of continuous landmark like a tomato crop. The reflections returned from a smooth corridor wall are extremely strong compared to reflections returned from common surfaces in nature (bushes, trees trunks etc.) where the signals are strongly attenuated thus leaving little energy for bouncing.

Experiments showed that the best sensing performance is achieved while the sensing line is almost perpendicular to the wall, due to the specular nature of the wall. Unfortunately, the high energy of the reflection can create a phenomena of multi-path reflection. This creates ghost images in the CTFM signature (or Acoustic Density Profile). The phenomena happen because of reflections bouncing between the wall and structure elements of the robot. It is like the generation of multi-images when looking between two parallel mirrors. The phenomena only occurred at a certain positions after coming too close to the wall and at a certain angels towards the wall.
Figure 9.2 Multiple-path reflection between the corridor wall and the robot's structure creates multiple images. The corridor wall is a very smooth surface that behaves like a mirror. The ultrasonic waves are bouncing back and forth between the sonar front plate or other smooth surfaces in the structure and the wall.

The result is several spikes at regular spacing in the acoustic density profile (Figure 9.2). A second consequence, is that a small turn of the robot can change the angle of the beam sufficiently to cause significant change in the acoustic density profile, as can be seen by comparing Figures 9.2 and 9.3.
Figure 9.3 At a certain shallow angle between the robot longitudinal axis and the corridor wall, the bouncing of the ultrasonic energy reminds the effect of looking between two mirrors with small inclination. The relative size of the peaks is fluctuating.

9.5 Detection of path edge and junctions in Wollongong Botanic Gardens

Experiments in path following in the Wollongong Botanic Gardens (Chapter 12.2) included the challenge of following the edge of a continuous path for 250 metres and coping with three junctions with other paths. The main paths in the garden are made of smooth concrete. There are junctions where a path divides into two or more concrete paths. The first and the second junction were of this type. Sometimes, the junction includes a diversion to an unsealed path in addition to a possible new smooth concrete path. The third junction had this mixture of path types.

In contrast to the corridor, where the surface is a sharp spike in the acoustic density profile, the acoustic density profile of the path edge (Figures 9.1 and 9.4) is quite complex. The first part has zero amplitude because the concrete reflects the energy away.
The second part is quite rough because the grass has many small reflectors pointing in many directions. The junction between the two is easy to detect.

Figure 9.4 Video shot taken from screen during navigation showing the signature of the path edge.

9.6 Effect of feed-forward looking on path edge

While driving a car, we tend to look forward to a point far ahead. This gives us advance knowledge of the expected edges of the road. Our brain transforms this data from vision to lateral angular deviation so we can respond by turning the steering wheel with fine corrections. Trying to look forward by using 45° pan angle instead of 90° reduced the amplitude of the echo from the grass because the energy reflected away from the path edge. The best navigation results were for the 90° pan angle. The reason is that the sonar sensing is gauging the distance to the edge when sensing perpendicular to the edge or the wall. The arm of the sensing line with respect to the mid-rear axis is also more effective (shown also in Chapter 2 - Figure 2.6).
9.7 LabVIEW programs for edge detection and following

The program FindFirstPeak3.vi implements the search algorithm that detects the first peak above a predefined threshold and concurrently checks the power level of the echo. From the location of this peak, the range difference to a predefined desired range (tracking error) are calculated. When the power level is drops beneath than a predefined threshold (tuned to the specific environment) the system assumes that its edge cannot be detected and uses a predefined range error. This predefined error puts the robot into a slow turn to the left toward where the edge is expected to be. The program is embedded as a sub-VI in SonarAnalyzer4.vi which is called by SonarCapture4.vi at each distance increment.

![FindFirstPeak3.vi Diagram](image)

**Figure 9.5** LabVIEW program for finding the edge of a smooth surface path with a rough surface verge.

To conduct an edge following leg, the designated sensing strategy is picked from the navigation map by the program in Figure 9.6. It is the top level of the continuous landmark navigation software and selects which sensory strategy is to be used in an experiment.
This program uses a sequence of frames to manipulate the appropriate VIs using dynamic calls.

Figure 9.6 LabVIEW navigation program showing the implementation of the strategy for following path edge (strategy 3 is specified in the navigation map for this mode). Various sensing strategies (SonarCaptureX.Vi) can be selected for different experiments.
Robot description

Titan is an outdoor mobile robot guided by CTFM sonar, developed from a 4wd wheel chair. Dr. Bernard Silderstein is a paraplegic who wanted a wheel chair that he could drive on sports fields, beaches etc. Mr. Jeffery Pagett designed and patented [Pagett, 1996] a new steering mechanism for a 4-wheel drive (4wd) wheel chair. This mechanism achieves differential velocity steering without skid steering. We have modified a commercial 4wd wheel chair (Titan) – into a 4wd out door mobile robot (Figure 10.1).

Figure 10.1. Titan 4 wheel-drive robot in various stages of development from a wheel chair.
10.1 Mobile robot design

Titan is driven by 4 wheels with pneumatic tires. Each wheel is mounted on the shaft of a gear motor powered by a pancake motor (Figure 10.2). The motors are emf controlled in pairs (one pair per side) by PID controllers. The user manually controls forward velocity and steering direction with a 2D joystick.

Titan is steered by controlling the differential velocity between the two pairs (one on each side) of wheels. Thus, the joystick steering input is added to the velocity on one side and subtracted from the velocity on the other. Normally, differential velocity steering of a 4 wheeled vehicle results in skid steering. This is not desirable for a wheel chair, because skid steering can damage carpets and lawn, and scuff cork floors and concrete.

Figure 10.2 Titan front drive system showing left pan cake motor and steering encoder.
An innovation in steering design, that has been patented [Pagett, 1996], enables differential velocity to achieve Ackerman steering. Ideally, an Ackerman steering mechanism turns the front steering wheels by different angles so that the axes of all 4 wheels pass through a common point. As a result, no sideways force is applied to the wheels and hence, no skidding. In practice, car steering mechanisms only achieve this ideal at one steering angle.

Like a car steering mechanism, the mechanism used in Titan only achieves Ackerman steering at one steering angle. The steering mechanism consists of 4-bar linkage. This design achieves precise Ackerman steering at one steering angle and generates some deviation at other steering angles. This mechanism was modelled geometrically using AutoCAD (Figure 10.3).

![AutoCAD model of steering showing deviation from ideal Ackerman steering](image)

Figure 10.3 AutoCAD model of steering showing deviation from ideal Ackerman steering

When the fourth link is open, both wheels are free to turn about the rotary joints connecting them to the chassis. This freedom is constrained by the forth link so that when one wheel turns the other also turns. When differential velocity is applied to the two pairs (one per side) of wheels, the front wheel with the lower velocity turns outward. The 4-bar linkage causes the front wheel on the outer side to turn with it, by a proportional amount. As a result, the configuration of the wheels approximates Ackerman steering. The robot turns in the
Geometric simulation illustrating the deviation from ideal Ackerman steering. The trigonometric manipulation involves the drawing of two circles and the finding of the location at the intersection. The diameter of the larger circle $c_1$ is equal to the length of the trapezoid roof. The diameter of the smaller circle $c_2$ is equal to the length of the trapezoid side members. This exploration is confirmed by Matlab simulation AckermanRadiusDeviation1.m illustrating the deviation from common turning center versus the inner wheel angle.
direction of the lower velocity wheels with a turning circle whose radius is inversely proportional to the differential velocity.

The geometric simulation in Figure 10.3 involves the drawing of two circles and finding the intersection point. The radius of the large circle is equal to the roof member of the steering trapeze while the radius of the smaller circle is equal to the side member of the trapeze. Firstly, the right wheel is rotated at $45^\circ$ followed by the construction of the larger circle centred on the roof joint of the rotated right side. The smaller circle is centred on the left front joint. The intersection determines the new angular position of the left side member coupled to the left wheel.

The gear motor for each front wheel is mounted on one of the 4 links (the two short links) as shown in Figure 10.4. These links are connected to the chassis by rotary joints, so the chassis forms the third (front) link. The fourth (back) link is an adjustable length rod that connects the motor links using rotary joints. The adjustment allows manual tuning of the angle of the front drive wheels so that they are parallel at zero degrees steering angle.

![Figure 10.4 Detailed view of 4-bar linkage mechanism for front steering wheels - the gear motors comprise two sides of the 4 bar linkage which enables skid-free differential velocity steering.](image)

An additional feature of the design is that the axes of the wheels do not intersect the axes of the rotary joints that connect them to the chassis. In fact, the wheel axes are behind the joint axes (Figure 10.4). The result is that, when the robot is moving in the forward direction, the front wheels castor. We believe that this design achieves a forward stabilizing effect,
making steering in the forward direction more positive. Unfortunately, the castor effect destabilizes steering in the reverse direction. Consequently, the joystick is physically constrained to only allow small steering angles in use.

![Graph showing non-linear behaviour of Ackerman steering](image)

**Figure 10.5** Non-linear behaviour of Ackerman steering

The non-linear transfer function relating the angles of the sides (on which the wheels are mounted) of a trapezoid were explored using a Matlab program. The 4-bar linkage was divided into two adjacent triangles (Figure 10.14) which were solved numerically. This Matlab function was also integrated into the dynamic model. Figure 10.5 shows the angle of the left wheel (upper graph) as a function of the angle of the right wheel. (lower graph).

### 10.2 Robot sensors and actuators

Figure 10.6(a and b) gives an overview of Titan's modular design. Figure 10.7 shows the electronic rack and the PCI expansion chassis. All the computing power is provided by a PowerBook G3. Data acquisition is achieved via a National Instruments general purpose data acquisition card (DAQ1200) a serial I/O card and a Frame Grabber in the PCI chassis (Figure 10.7 right).
Figure 10.6a General view of Titan’s electronics, actuators and sensors during autonomous navigation in Wollongong Botanic Gardens
Figure 10.6b Architecture of Titan’s electronics, actuators and sensors

The two rear encoders measure the travelled distance (Figure 10.8). The steering encoder (mounted on the left front joint) provides steering angle for the left front wheel. The gyroscopic stabilised digital compass measures the magnetic bearing (Figure 10.9). We are using a KVH Digital Gyro Compass which was originally design for yachts auto-pilots. Titan's rate of turn can be much faster than a yacht and hence the real-time difficulties of using it to control the magnetic bearing especially at the transitions between legs.
Figure 10.7 Electronic rack (left) and PCI rack and instrument control boxes (right)

Figure 10.8 Mounting of position encoders on rear wheels
The robot is equipped with a custom [Kay, 2000] built CTFM (Continuous Transmission Frequency Modulation) sonar (Figure 10.10) which continuously transmits chirps with a linearly decreasing frequency (100kHz to 50 kHz at 100 msec duration). The transmitter is a 20 cell phase array while the receiver consists of 4 single cells located on the 4 surfaces of a shallow pyramid. The CTFM sonar is mounted on a pan & tilt unit.
10.3 Robot software

Robotics research is fascinating because biological systems provide the inspiration for the robotic systems we are attempting to develop. The brain of the biological systems is often emulated using software in the robotic system. However, the time consumed to develop this software may delay the research. This software must be both efficient to use and to develop. The purpose of high-level languages and engineering tools is to facilitate the design and development of software, so that developers can quickly create new systems to test research ideas.

Traditionally, robot software is built by writing lots of C/C++ code. This requires programmers with professional expertise in C/C++, real-time operating systems, interface drivers and robot control. A lot of time is consumed developing software infrastructure prior to any tangible research effort takes place. In this paper, we discuss an alternate approach to engineering a research robot using commercially available software tools. The design/development process is broken down into stages and a separate tool is used for each stage. Stage 1 is the modeling and simulation of the kinematics of the robot using AutoCAD. Stage 2 is the modeling and simulation of the control and ultrasonic sensing system using Matlab and Simulink. Stage 3 is the implementation of the control and sensing software using LabVIEW. In this research, we illustrate the use of these engineering tools through the design of Titan.

LabVIEW is a visual programming system specifically designed for data acquisition, including drivers for interface cards. Each program module is a virtual instrument with a front panel (Figure 10.12) and a circuit/program diagram (Figure 10.11). LabVIEW has tools for constructing the front panel controls and displays and for programming the block diagram using functions.

The block diagram in Figure 10.11 shows a program for continuously reading an analog input and displaying it on a chart. The analog input software is a sub virtual
instrument (a nested function call) shown as a square, and the while loop is the large outer rounded square. A toggle switch is used to control loop termination.

**Figure 10.11** LabVIEW block diagram - While loop containing a single analog input.

**Figure 10.12** Virtual front panel for testing the sensors and actuators of the Titan mobile robot displayed on a Macintosh PowerBook
Figure 10.13 Virtual front panel for control of Titan mobile robot

A researcher can control the Titan mobile robot from a virtual front panel (Figure 10.13) on a Macintosh PowerBook. The PowerBook is interfaced to the robot controller and to the sensors using PCI cards. The PowerBook is a simple way of getting on board computing. As designed, the person sitting in the wheelchair controls it with a 2-dimensional joystick. In this manual mode, the computer displays the joystick values for velocity and steering on the chart as the user drives the vehicle.

With this front panel, the user can select two modes of computer control: joystick or slider. In joystick mode the computer reads the joystick commands and after modification outputs them to the controller. This mode is suitable for the teaching of robot paths. In the slider mode, the computer outputs the values on the virtual sliders to the controller. This mode is used to manually set velocities for modelling experiments and will be replaced by autonomous control for the landmark based navigation of the robot.

The LabVIEW program for the robot control front panel took less than 2 weeks to develop including learning LabVIEW and testing the interface from the robot controller to the PCI card.
10.4 How distance support works

As Matlab/Simulink and LabVIEW are not taught by the school of IT & CS at the University of Wollongong, technical support is largely facilitated by e-mail communication with the relevant support groups in the USA. The help gained via these interactions was very valuable. We found that technical consultation and support via the Internet could be very professional and it enabled robotics research to be conducted efficiently.

In contrast, in the traditional approach all information is gained locally, and the researcher spends much time finding answers to technical details. It takes time to think "globally" and not to rush to the person in the next office, which many times just interrupts a colleague in his work and does not provide a solution. However, it takes time to learn how to work efficiently with support people over the Internet. You have to present them with a clear definition of your problem and provide them with the complete information they need to analyze your case.

On sending an e-mail enquiry to the support groups in the USA, the service acknowledges the request by sending an automatic e-mail message confirming the request and issuing a reference number for the enquire. The technical data regarding the problem was transferred by three methods:

- Attachment of a file with the model (Matlab/Simulink or LabVIEW) to the e-mail (we use Netscape 4.5 mail, set to the MIME method). LabVIEW 5.0 enables you to save your model in a library folder which contains all the code pieces in the virtual instrument hierarchy tree, saving you the difficulty of collecting all the pieces (and avoiding missing a small file which forces a new round of correspondence). Matlab 5 and Simulink 2 do not have this facility.

- Copying error messages from the Matlab control console and pasting them into the e-mail message.
• Attachment of JPEG images of the screen dumps or diagnostic windows. This was done by converting the Macintosh PICT format to JPEG format, so that it could be read on any machine.

• Attaching images is very helpful when you get an error message which can not be copied and pasted in the e-mail. It is especially important in the case of real time software which operates hardware (like CTFM sonar or a digital gyro compass) to which the support group has no access.

In the case of LabVIEW, images of the screen can show to the support group the output of project debugging tools. For example, the 'Hierarchy Window' which shows the relationships between the user defined modules and library blocks. The 'Profile Window' can be used to analyze real-time software. It shows the distribution of computational resources, timing statistics and memory usage. These project tools in LabVIEW enable remote experts to suggest diagnosis.

10.5 Analysis of kinematics with AutoCAD

We used AutoCAD to investigate the geometry of Titan kinematics (Figure 10.3). Titan has a unique Ackerman steering system is using a floating 4-bar linkage (Figure 10.4). This unique design provides skid-free steering using differential velocity command from a single joystick. The system is discussed in detail in a paper devoted to the kinematics and dynamics of the Titan mobile robot [Ratner & McKerrow, 1999].

Also, we used AutoCAD to clarify the phenomena - which was found in the Simulink analysis - of a dynamic cross-coupling between the steering angle and the contribution of the front drive system to the turning moments (Figure 10.15). During the transient of a right hand turn, the moment arm of the left front wheel is increased with respect to its value before turning and the moment arm of the right front wheel is decreased. This generates a momentary increase of moment-steering coupling. The driving forces are shifted from the contact point of the tyres to the front joints according to Figure 10.19. The residual moments due to the shift
and RadiusSizeControlr1 (and also for the Simulink dynamic & navigation model TiitianNavigationModel2ml).

Figure No. 10.14
of the forces to the joints are balanced on the floating steering system stabilised on a no
skidding pose.

![Figure 10.15](image1.png)

**Figure 10.15** Effective moment arms of front wheels due to Ackerman steering.

### 10.6 Modeling kinematics with Matlab

Next, we modeled the relationship between the steering angles of the left and right
wheels (Figure 10.3). The 4-bar linkage was modeled as two adjacent triangles (Figure 10.14)
in Matlab. The output of this model in Figure 10.16 shows the non-linear relationship
between the steering angles. The Matlab program was later transformed into a Matlab function
and embedded in the S-Function block responsible for calculating the contribution of the front
drive system to the turning moments for the robot in the Simulink model.
Figure 10.16 Non-linear behavior of Ackerman steering. Lower graph shows linear change of right front wheel from zero to $-45^\circ$. Upper graph shows the steering angle of the left front wheel from 0 to $32^\circ$.

The kinematics equations of a 4 wheel robot with floating front drive Ackerman steering (Figure 10.6) are similar to those of a turtle robot [McKerrow, 1999]. In a turtle robot, the robot frame is located at the center of the robot. The motion of a turtle robot can be described with three modes: pure rotation, pure translation and a combination of rotation and translation. In the case of Titan, the robot frame is located mid way between the rear wheels. Because of the angular limitation of the steering joints, the nonholonomic robot has only two modes of motion: pure translation, and combined translation and rotation [Sekhavat and others, 1998].

The angular velocity of the robot (Equation 10.1) is found by dividing the differential linear velocity of the rear wheels by the rear wheel span. The turning radius (Equation 10.2) is found by dividing the linear velocity of the robot by the robot angular velocity.

Figure 10.17 Kinematics model
\[ \omega_R = (V_f - V_r) / W_R \]
\[ r_{rot} = 0.5 \cdot (V_f + V_r) / \omega_R \]

10.7 Modeling dynamics with Simulink

A Simulink model (Figure 10.18) was developed to explore the dynamic features of the robot. The model helps to illustrate the effect of the floating Ackerman steering system on the motion of the robot. The cross coupling of the steering angles and the driving torque from the front drive wheels were investigated and later clarified using AutoCAD schematic in Figure 10.15.

Another incentive to develop the model was to use it as a test bed for designing and testing algorithms of the recognition-based navigation. We found that simulation can help to reduce the amount of outdoor experiments. The model was implemented using 5 special user defined S-Function blocks (written in Matlab code) in addition to the standard library blocks.

![Simulink block diagram model for Titan's dynamics and early stage of development for navigation. See also navigation model (Figure 2.37) and simulations in section 2.17.](image)
The simulation of the navigation (Figure 10.20) is realised in the Simulink model by introducing of a S-Function block which reads the planned trajectory from a map and compares the current position of the robot with respect to the landmark to the designated position. The error is used to calculate new directions and distances to the next landmark.

The dynamics of Titan is best explained by referring to the a Simulink model shown as a block diagram in Figure 10.18 and the AutoCAD schematic for acting forces on the model in Figure 10.19. The dynamic model is based on the Newton-Euler formulation of dynamics. Simulink is a graphical programming language for control system modeling using block diagrams. To enable the modeler to include complex world models Simulink allows the user to program an algorithm in Matlab and define it as a Simulink S-Function block, which can be used in the block diagram model.

The model in Figure 10.18 includes two parallel loops: one for calculating linear motion (top) and one for calculating angular motion. This decomposition matches the inputs to the robot's controller. These velocities are calculated from the forces applied to the wheels.

The linear motion loop (top) calculates the linear position and velocity of the robot for the forces applied to the robot, less the resistive force due to friction and damping. The input to the loop is an applied force calculated by S-Function 1 from the torques applied to the wheels. These wheel torques are calculated by S-Function 3 to achieve the controlled trajectory.
Figure 10.21 Simulated trajectory following with increasing loop gains (top to bottom), showing change in damping.
The angular motion loop is similar to the linear motion loop. Its input is a turning moment calculated by S-Function 2 from the torque applied to the wheels. On the right hand side of Figure 10.18, the x,y position of the robot is calculated from the motion values. At the bottom of the diagram, the turning radius and wheel angles are calculated for display.

The forces and torque applied to the robot are calculated in the S-Function blocks. The Torque Motors Control (S-Function 3) block reads the trajectory from a look up table. The table contains a distance and a bearing for each leg of the trajectory. It calculates the current location of the robot from the outputs of the above loops. From the trajectory and current location it calculates the error in reading. The output of this block is a differential torque proportional to heading error which is clamped to model the limitations of the motor control electronics of the robot.

The forward velocity (S-Function 1) block calculates the forces applied to the robot from the differential torque and the angle of the steering wheels. The angle of the steering wheels is calculated from the differential velocity between the two sides of the robot.

The Turning Moment (S-Function 2) block calculates the turning moment applied to the robot from the differential torque and steering angle. This system implements a Newton-Euler formulation because it calculates the acceleration (angular and linear) in each loop from the balance of the forces on the robot. The mass of the robot is approximately 100 kg.

Simplifying assumptions are used in the model. First, we assumed that the friction forces are large enough to avoid lateral sliding of the wheels. Second, we assume that the friction force resisting the turning of the wheels (and hence producing forward motion without skidding) is adequately modeled by the Simulink Coulomb and Viscous Friction block. Thirdly, we assume that the friction in the joints of the steering is negligible and the inertia of the links in the 4-bar linkage is small compared to the inertia of the robot.

The output from the simulation for a trajectory consisting of a right turn (from 100° heading to 190°) followed by left turn (from 190 degrees heading to 100 degrees) is shown in Figure 10.21. As the loop gain is increased, the response moves from overdamped to
underdamped. The following three sub-figures (Figure 10.21) describe few behavior patterns as part of the sensitivity tests for parameters like the proportional control gain.

10.7.1 Steering angle coupling

The simulations show a coupling effect between the turning moment contributed by the front drive wheels and the steering angle. In Figure 10.22, the turning moment produced by the front wheels is plotted against time for a 90° turn. The upper curve is the turning moment when the steering wheels are free to turn. The lower curve is the turning moment when the steering wheels are locked at zero steering angle. From these curves, we see that turning the front wheels during a turn increases the turning moment produced by the front wheels even though it reduces the motor torque.

![Figure 10.22 Effect of Ackerman steering on the front drive turning moment. The upper curve is the turning moment delivered by the front drive. The lower curve is for zero steering. Both are from the clamped controller.](image)

The effect of increased moment from reduced torque occurs because the effective moment arm of the outer wheel increases (from 185 mm to 316 mm) and the effective moment arm of the inner wheel decreases (from 185 mm to 27 mm) in Figure 10.15.

10.7.2 Robot experiments

Titan's sensors and actuators are connected to a Macintosh PowerBook with National Instruments PCMCIA data acquisition card. The sensing and control software is written in
LabVIEW [Ratner and McKerrow, 1997]. A virtual control panel (Figure 10.23) was built to control this experiment. The location and heading of the robot were measured using encoders attached to the rear wheels and a gyro stabilized compass. The steering angle of the left front wheel was measured with an encoder.

In order to compare the motion of the robot with that predicted by the simulation, we developed a feedback loop to control vehicle heading with a LabVIEW formula node. The robot was driven along the same trajectory as used in the simulation. A trajectory is specified as a sequence of tuples. Each tuple specifies a heading and the distance to be traveled in that direction. The trajectory in Figure 10.21 is 10m at 100°, 7.5m at 190°, and 7.5m at 100°. The linear velocity of the robot is fixed for the length of the trajectory, and the differential velocity is controlled with the feedback loop.

Figure 10.23  LabVIEW virtual control panel for the experimental set up
Figure 10.24 Trajectory of an outdoor experimental run at early stage of investigation. The iterations of simulations and experiments contributed to improvements of the robot’s motion control and refinement to the Simulink model. The navigation simulations in Chapter 2.17 demonstrate very good results in using the simulations to develop the navigation methodology.

We studied the dynamic behavior of Titan by varying the loop gain and linear velocity. We found that the experimental response was similar to that of the simulation. Both transferred from overdamped to underdamped response on bearing changes as the loop gain was increased.

However, the actual trajectory in Figure 10.24 is different to the commanded trajectory. First, the robot must follow a curved trajectory to turn from one bearing to another. This results in a tracking error. Second, the experiment shows that the robot is slowly creeping to the right on the straight-line segments. This is due to a mismatch in the zero volt line between instrumentation on the robot and was later investigated and fixed by changing the electronic design.

Thirdly, the right turn is much sharper and less damped than the left turn. This may be due to the mismatch mentioned above. However, we have not confirmed this.

10.7.3 Conclusion

The experiments show that the Simulink model is a reasonable representation of the robot dynamics. To model this robot we have used three complementary tools: AutoCAD for
geometric modelling of kinematics, Simulink for block-diagram modelling of the dynamics, and LabVIEW for controlling the robot. This combination has enabled us to be very productive.

10.8 Programming the robot controller with LabVIEW

The LabVIEW graphical language provides the user with a rich engineering environment of software tools for instrumentation called virtual instruments (VI's). The code of the tools is not visible to the user (like in Matlab), but there are many project tools which are vital when building fast prototype software for a complex project like a mobile robot with sensor suit. To control the Titan robot (Figure 10.25) several virtual instruments for sonar, compass, encoders, pan&tilt mechanism, navigation etc. work in parallel and interact with the hardware (data-acquisition) of the robot and sensors.

![Virtual control panel built with LabVIEW](image)

**Figure 10.25** Virtual control panel built with LabVIEW drag and drop elements and glued together with graphical language.

LabVIEW uses global variables to link information between the VI's. The engineering experience of National Instruments is reflected in the tools and the on line documentation. These features enable anyone with an engineering background to handle the huge amount of software involved in projects of an unmanned vehicle system.
10.9 LabVIEW compatibility with Matlab

As mentioned, most of the software development process was based on repeated cycles of design iterations in the Matlab/Simulink environment and implementing the system in a real-time environment with LabVIEW for testing. LabVIEW has a Code Interface Node tool (CIN) which enables the linking of external C/C++ code (all LabVIEW library functions - called VI's - are written as CIN, so their compiled code is not transparent to the user). We thought that we might be able to create CIN's by including the C/C++ code generated by a Matlab compiler (we used Matcom by Mathtools to try it). LabVIEW generates a template C/C++ file automatically from a front panel which contains the input controllers and output indicators. Unfortunately, the technical formats (data type definition etc.) do not match. If this problem could be solved, it would enable the development of complicated algorithms in Matlab and testing them immediately on the robot.

Running compiled C code with CIN is very time efficient and is important for tasks such as recognition algorithms which include statistical processors like spectrum estimators, cross-correlation and feature extraction.

10.10 The LabVIEW Development Environment

The use of LabVIEW to control complex mobile robot was very challenging for a mechanical engineer without the traditional background of C++ since it proved the feasibility of using engineering software tools for programming complex robotic system. This software is not taught in the university (as well as Matlab/Simulink and AutoCAD) and most of the technical support was provided over the Internet and some by the local agent in Australia. Using LabVIEW enabled us to channel our efforts efficiently in solving robotics issues and much less being engaged in pure technical hurdles. Creativity could be expressed more easily and tested on the fly. The navigation program for Titan is using extensively dynamic calls to manipulate many other VI's (library and user defined Vis) and thus enabling modularity and transparency for a complex structure which shrinks and expands during run time.
The LabVIEW software developed during the 5 years of the Titan project was later a base for developing a library of reusable components by David Skoumbourdis "LabVIEW Library for Mobile Robot Control" [Skoumbourdis, 2000]. His CSCI401 Computing Science IV Honours Thesis details the algorithms for interfacing the sensor suits.

When running the navigation program (depends on the chosen navigation strategy) and using the project tool profile window – a total of 100 files (VIs) are involved (around 50 user-defined files and around 50 LabVIEW library VIs). The profile tool shows also the distribution of memory consumption and number of calls to each VI. Another (in addition to many other features for prototypes development) useful project tool is the hierarchy window showing the relationships between the programs and thus supporting the modularity of LabVIEW. Using dynamic calls helps to create compact structure for the master navigation program (NavigateCombinedStrategies2.vi) but these calls (during run time) are missed representation in the Hierarchy Window which shows the included VIs only when the program is opened.

The following three figures (Figures 10.26 to 10.28) illustrate the amount of code involved in the navigation program. Figure 10.26 shows the block diagram of the master navigation program NavigateCombinedStrategies2.vi. The main program is set on the option of navigation between isolated landmarks using recognition with angular sweep. Figures 10.27 and 10.28 show run time statistics produced by the profile window included in the project tools. The data in this example relates to one leg of edge following in the lab.
Figure 10.26 The block diagram of the master navigation program to manipulate the navigation and sensing strategies and also the motion control. Its compact appearance includes layers and frames to accommodate the behaviour dictated by the navigation map. Dynamic calls are used extensively. The figure shows the main program set to the option (strategy 1) of navigating between isolated landmarks with landmark recognition based on angular sweep (Algorithm in Appendix B.2).
Figure 10.27 Profile window showing the status of number of calls and memory usage for the user defined programs (VIIs) and library programs (VIIs) during execution of the navigation program. The statistics helps to identify bottle necks and optimise the usage of CPU resources in multi tasking.
Figure 10.28 Expended VI data provide additional details of the activity associated with each of the 100 VIs included the list of Figure 10.27. Compare (for example) line 8 in Figure 10.27 for SonarAnalyzer4.vi with line 110 in this figure for SonarAnalyzer4.vi.
10.11 LabVIEW software architecture for Titan control

During the 5 years of the research, a large amount of software was written. This includes all the interfaces to the sensors and actuators, motion control loops and navigation algorithms. This section explains the software architecture (Figure 10.29) used for Titan control in the master navigation program (NavigateCombinedStrategies2.vi) (Figure 10.26).

10.11.1 Selection of navigation mode
At the top level, the master navigation program (Section 10.11.2) selects the navigation strategy for the next leg of the current experiment from information in the map. The map includes a strategy flag in the description of each leg. It can choose from one of four strategies at the start of each leg.

1. Without sensing - This mode is used for teaching the navigation map including automatic recording of reference signatures. It also illustrates odometry problems. The mode is equivalent to Instrument Flight Rules.
2. With angular sweep - This mode is used for recognition of isolated landmarks. This is the equivalent to Visual flight Rules.
3. With linear sweep - This mode was used for testing the feasibility of recognising isolated landmarks from a moving robot.
4. Edge following - This mode is used for following walls and paths and is equivalent to coast line following.

10.11.2 Software for Visual Flight Rules strategy
As an example, we will explain strategy 2 in the previous list (flag == 1 in the navigation map and the navigation program in Figure 10.26) the strategy used for navigation with isolate landmarks. The master program selects option 1 (Figure 10.26), which schedules a sequence of steps to achieve this strategy. Those steps are:

1. Initialises the I/O cards and global variables.
2. Moves the pan & tilt unit radial sensing position.
Navigation Program

1. Initialisation and strategy selection

2. Main navigation program
   A. Figure 10.26
   B. Figure 10.26

- Sonar sensing
  - 10.11.3.1

- Moving along a leg
  - 10.11.3.2

**Setup VIs**
- 1. Pan & Tilt controls
- 2. Navigation map
- 3. Data recorder

**Files**
- Recorded files

**Sonar VIs**
- 1. Sonar signature
- 2. Angular sweep control
- 3. Navigation correction calculation

**Files**
- Reference signatures

**Motion acquisition VIs**
- 1. Position encoders
- 2. Odometry calculation
- 3. Velocity calculations
- 4. Compass
- 5. Inclinometer

**Motion control VIs**
- 1. velocity PID loop
- 2. Steering PID loop
- 3. Bearing PID loop

- 1. Distance control
- 3. Turning control
3. Starts the navigation map and data recorder.

4. Runs the main navigation program (shown as frame 8 in figure 10.26) which starts the VIs to achieve the visual flight rules navigation strategy (Section 10.11.3).

5. When the experiment is over, it terminates.

### 10.11.3 Main Navigation Program

The main navigation program implements the desired navigation strategy. In this case, the visual flight rules strategy is used to navigate the path between two poles. The main navigation program repeatedly performs a two step sequence of operations.

1. Sensing step - sweep the sonar to scan the pole and find it.
2. Motion step - move the robot to the next pole using the corrected map.
3. Repeat the above two steps.

#### 10.11.3.1 Sensing the landmark

The sensing step is performed by calling 3 VIs (which call many Sub-VIs): Pan & Tilt, Sonar Signature and Angular Sweep. These localise the robot and calculate the new parameters for the next leg of the map using the following steps.

1. Set sensor tilt position to the angle defined in the navigation map, and pan to start angle.
2. Clear leg distance and leg bearing in map.
3. Starts sonar data acquisition and conduct angular sweep.
4. Reads a sequence of 51 scans.
5. At the end of data acquisition, it finds the best match and pans the sonar to that angle, to confirm the match and measure the range and angle.
6. Calculates the new leg parameters and stores them in the map.
7. Terminates sonar data capture.
10.11.3.2  Moving along the leg

The motion step is performed by calling 11 VIs (which calls many Sub-Vis). These VIs can be classed in two groups: data acquisition and control.

The data acquisition VIs perform two functions:

1. read wheel motion and steering encoders and calculate position, direction and velocity of the robot using odometry equations, and
2. read the compass and calculate the heading of the robot.

The motion control VIs use the data as feedback to move the robot along the leg specified in the map, including straights and curves. They include:

1. PID control loops for velocity, steering and bearing; and
2. control algorithms for moving a target distance and turning around a curve with a target radius.

During each leg, the Vis required for sensing and motion are called dynamically as they are required. Using dynamic calls has the advantage that every thing (encoders counters, controllers etc.) is reset and no accumulation of errors propagates from one leg to the next leg. It also helps to reduce the load on the computer by closing unnecessary VIs (separation of motion from recognition). From the programming point of view it enables modular and neat programs. The only disadvantage is that VIs which are dynamically called are not automatically represented in the Hierarchy Window, which is important for the saving option of a development library. In this option, all the user-defined files are saved into one library that could be attached to e-mail for consultation.
Figure 10.30 Top figure shows activation of the distance controller reading the corrected navigation parameters. Bottom figure shows terminating the group of 8 motion control VIs in Figure 10.26 at the end of the landmark leg (Algorithm in Appendix B.2).
10.12 Conclusion

To develop the software for this robot we have used three complementary tools: AutoCAD for geometric modelling of kinematics, Matlab for kinematics modelling and sonar analysis, Simulink for block-diagram modelling of the dynamics, and LabVIEW for controlling the sonar and robot. This combination has enabled us to be very productive.

Also, we have found that the technical support over the Internet for Matlab/Simulink and LabVIEW is very helpful as models are fully portable by e-mail attachments. The Internet provides the capability for very efficient consultation and opens new opportunities for creative engineers to contribute to fast prototyping for robots and to algorithms for real-time systems.

The advantages of this approach are:

- software can be written more quickly, leaving more time for research,
- you do not have to be a computer scientist to program the system, and
- help is readily available from the tool supplier over the Internet.
11 Motion control

Good quality motion control (velocity control, distance control, angular distance turning and size of the turning radius) is an essential feature for Titan's navigation capability in outdoor environment to provide accurate approach to landmarks [Ratner and McKerrow, 2000c]. Titan has to follow a navigation map, which consist of a sequence of legs. A leg can be straight, curved or the edge of a path.

The control strategy used for following the edge of a continuous landmark (Chapter 2.3) is different to that used when following a sequence of straight legs (Figure 2.1). When following a continuous landmark the control system must track the edge of the path and control the steering to maintain the robot at the correct tracking distance from the edge. The architecture of the edge following control is shown in Figure 11.1.

The controller consists of two parallel outer control loops: range and distance. The range controller compares the measured range to the edge of the landmark (path, wall, etc) to the target tracking range to calculate both tracking error and rate of change of tracking error. A proportional plus derivative (PD) controller is used in this loop. The distance controller is rule based because a traditional proportional plus integral plus derivative (PID) controller requires feed forward control (as well as feedback) based on an accurate model of the robot’s dynamics to achieve stable and accurate control (Section 11.5).

In contrast, when travelling a long and straight leg between isolated landmarks, the control system must maintain a constant bearing (Figure 11.2). The controller consists of two parallel outer control loops: range and bearing. The bearing controller compares the bearing measured with a compass to the target bearing to calculate tracking error. A PID controller is used in this loop. The distance controller is the same as for following a continuous landmark.

However, turning the robot from one bearing to another around a curve of any given radius requires a hybrid strategy to control bearing (Figure 11.3). We found that a PID loop oscillated badly when attempting to make the large changes in bearing required by planned turns. So a PID loop is used when tracking along the straight portion of a leg. When a turn
At the start of each leg by dynamic controlling to avoid accumulation of error from the previous leg. A flow of range data (taken every 0.25 meter) provides direction along the path edge. All motion control VIs are reset Titan's motion control block diagram for edge following legs. Note that the compass is not used and instead the
By dynamic calls to avoid accumulation of errors from the previous legs.

Titan’s hybrid motion control block diagram for landmark legs and blocking legs. All motion control VIs are reset at start of each leg.
Bearing controller for landmark legs comprises of an hybrid PID and rule based controller.
is required the hybrid controller switches to a rule based strategy, similar to the rule based distance control strategy, for the duration of the turn (Section 11.6).

A leg starts with a turn to a new bearing and proceeds with straight line section. In the isolated landmark case, it uses the capability to recognize landmarks to correct the map at the start of each leg. The motion along the leg (between the isolated landmarks) is controlled using bearing data from the gyro-stabilised compass and odometry data from the two rear encoders. The steering is controlled using a steering PID with feedback from the steering encoder. In the continuous landmark case, the recognition of the path edge is used to determine the range error as input for the motion controllers (the compass is not used for motion control).

Titan has unique kinematics and dynamics due to its floating Ackerman steering system (combined with castering effect aimed to stabilise forward motion) and its 4WD power train system. Titan achieves steering by differential velocity without skidding. These features make motion control of more difficult than for two wheel robots or four wheeled vehicles with a steering wheel. The robot can not turn on the spot (like a two wheeled robot). Also, it is quite sensitive to back driven steering since there is no gear box friction to provide resistance to (like in cars) the road's disturbances. A new concept of the “time to contact” (or “distance to top”) was used to improve the turn and distance control.

11.1 Steering controller

11.1.1 Geometric simulations for Titan’s steering

Figures 10.3 and 10.4 show the four-bar linkage of the front wheel steering system. Section 10.5 and Figure 10.14 give the kinematic analysis of the floating Ackerman steering of Titan. In this section, we examine the effect of this kinematic design on the control of steering. We found that we had to close the loop on steering with a very stiff controller, because any bump will back drive the steering wheel causing the robot to veer from a commanded path. To measure the angle of the front wheels for steering, we mounted an
There is only one singular configuration and the rest deviates from common center. The approximation as implemented in Radius_Sim steer is simulated cart. and implemented as Sub-1 in the front joint. The turn controller V.1. turns the steering wheel to have a common turning point. In Ackerman steering model, the steering angle of the left wheel is mounted on the left front wheel, which is the angle seal of the steering encoder which is mounted on the left front wheel.
encoder on the joint that the left front wheel rotates around. The relationship between the steering angle of the robot and the turning radius is given in Figure 11.4. Note, that we are measuring and controlling the angle of the left wheel.

11.1.2 Matlab simulation for steering kinematics

To explore the features of the robot's kinematics we used AutoCAD for geometric simulations and Matlab for numeric simulations. Titan’s steering system is a unique hybrid of a normal car with a 4 bar linkage actuated by steering wheel and a tracked vehicle’s differential velocity steering. Titan does not have a steering wheel, the four bar linkage moves in response to the forces on the wheels and maintains a mechanical relationship between the wheels.

![Figure 11.5](image)

**Figure 11.5** Matlab simulation (left plot) for the turning radius of the two wheels versus the steering angle of the inner wheel (Figure 10.3). The right plot depicts the difference in the steering radius between the two wheels versus the steering angle.

For true Ackerman steering, the axes of all wheels should intersect at a point. In Figure 10.3, we see that the axes of the steering wheels both intersect the common axis of the rear wheels, but not at the same point. The distance between these intersection points varies with the steering angle) as shown in the simulation in Figure 11.5. The steering radius of each wheel is the distance from these points to the centre of the robot (mid rear axis).

A second simulation (Figure 11.6) shows the progressive movement of the 4 bar linkage as the steering turns from 0° to 45° to the right as in Figure 10.3. The steering angle
of the two wheels, for this sequence, is plotted in Figure 11.7. The middle plot is the average of the angles of the two steering wheels and is represented in Figure 11.6 with an additional line in the middle of each diagram.

We used the result of this simulation to produce the geometric approximation (Figure 11.5a) of the steering system shown in Figure 11.4. From this approximation, we developed a VI (Figure 11.37) to calculate steering angle of the left wheel required for the robot to track around a curve with a specified turning radius. Hence, the navigation map specifies a turning radius and the control system calculates the reference for the steering controller.

![Matlab code performing the approximation in Figure 11.14 for the steering angle at the left front joint corresponding to the turning radius dictated by the navigation map.](image)

**Figure 11.5a** Matlab code performing the approximation in Figure 11.14 for the steering angle at the left front joint corresponding to the turning radius dictated by the navigation map.
Figure 11.6 Matlab numeric simulation showing a sequence of 18 positions of the steering system when turning to 45° to the right. The middle bar (which does not exist) represents the average steering angle.
Figure 11.7 Matlab numeric simulation correlation between the right and left wheel angels corresponding to the sequence of movements depicted in the previous figure.
11.1.3 Virtual control panel and preliminary driving tests

The virtual control panel (Figure 11.8) provides a few helpful functions and was used extensively during the research. The panel enables the monitoring of the manual joy stick commands, and emulates the joystick by two sliders when the robot is driven by the computer. We used it in an early research phase to explore Titan’s kinematics performance while driving the robot with the sliders. We attached to the robot a vertical stick with white chalk attached to its end. The sliders set the forward velocity and the different velocity to control the turning radius.

Figure 11.9 shows sets of circles of a tight turn on sealed bitumen plot, which overlap with nice consistency. The resulting tested circles plotted with variety of slider setting demonstrated Titan’s stability with a load of a person on board. The panel was also used when developing the autonomous modes to study the control problems. During the planning phase of navigation map experiments, the virtual control panel was used to monitor distances and magnetic bearings, when a computerised landmark map of the site was not available.

Another useful facility is the capability to record sessions of joystick commands and then play them back. This was used to study the repeatability of the control.
Figure 11.8 Titan virtual control panel was used in preliminary explorations of the platform stability.
11.1.4 Steering controller design

The steering controller VI (Steering&VelocityController3.vi) is a PI controller closing the loop on the angle of the steering encoder (which is installed on the left joint of the front drive system). The outputs of the steering and the velocity is conveyed to two channels of analog output using AO Update Channel.VI. These channels drive the original wheel chair amplifiers (made by Dynamic, NZ). This design creates the interface between the software and the hardware.

A PID controller is achieved with a formula node (Figure 11.10). For steering control only proportional plus integral gain was used to obtain a stiff control loop. The loop reads the reference from the bearing controller, feedback from the steering encoder and calculates a new output every 50 m/sec.
11.2 Velocity control

Titan's velocity is controlled by a clamped PID controller (Figure 11.11) which takes its feedback from the robot's velocity. The velocity is the average of the two rear wheels velocity readings (this average is the linear velocity of the mid rear axle). The reference to the velocity controller is the trapezoidal shape velocity command that is generated by the rule based distance controller. The output of the velocity controller is relayed to the Steering&VelocityController3.vi via a global variable where it is written to the analog output.

Measuring the velocity is somewhat impaired by the large diameter of the encoder coupling wheel which results in an almost 1:1 ratio (due to the need to add the encoders rather than integrate them in the DC servo motors like in the Yamambica or LabMate robots).
between the 2500 Pulses/Rev encoders and the rear wheels. Despite these design constraints, the robot exhibits smooth and accurate motion. To calculate velocity, the position encoder readings and a time stamp are recorded every 100 milliseconds. From there an incremental distance and an incremental time are calculated. The time stamp is needed because labVIEW is soft real-time and can not guarantee reading at exactly 100 millisecond intervals. The impact of the course velocity measurement on position control is discussed in section 11.5.

Due to the inertia of the robot, noise in the velocity measurement is smoothed out. The controller achieves quite smooth velocity control except at low velocities.
Figure 11.11 Titan's velocity PI controller is using as input the trapezoidal pattern of the reference velocity generated by the rule based distance controller. Algorithm in Appendix B.4.
11.3 Bearing controller

Bearing is controlled using feedback of magnetic bearing from a gyro stabilised compass. The compass is very stable in an outdoor environment but does not update as fast as the robot turns on a large turn. It is rendered useless in the laboratory by the steel reinforcement in the concrete walls and floor.

Also, when turning through 0° (North) the output of the compass switches from maximum to minimum. If not corrected, this can result in apparent positive feedback around 0°. The equations in the upper formula node in Figure 11.12 wrap the bearing error around to correct for this.

As mentioned in the introduction for this chapter, we explored a few approaches to implementing bearing control. The solution presented here is based on an hybrid controller which uses PID to implement tight following for straight line section and switches to a rule based controller to perform the turns with a prescribed turning radius from one bearing to another.

The bearing PID loop was relative easy to stabilise. As can be seen from Figure 11.13, the robot follows a fixed bearing with reasonable accuracy. In figure 11.13, the small circles are at the centre of the planned turning circle. When changing bearing by 90°, the PID controller overshoots. Worse, there is no control over turning angle, so the robot regularly ends up on a path parallel to the desired path.

Figure 11.14 shows the command bearings for the rectangular spiral path in Figure 11.13. The compass reading (curve with spikes in Figure 11.14) shows how far the bearing overshoots at 90° turn- roughly 60° or 54°. In section 11.6, we develop a controller that performs much better at turns.
Figure 11.12 Bearing PID loop. The upper part initiates a dynamic call to the turn controller if bearing error is greater than $15^\circ$. The upper formula node transforms the range of the bearing error to plus minus $180^\circ$. Algorithm in Appendix B.5.
Figure 11.13 Measured bearing control on an open field (without the introduction of the call to activate the turn controller). Reference trajectory indicated by short lines with circles at the start of each turn.
Figure 11.14 Bearing changes (commanded and measured) when following path in Figures 11.13.
11.4 Range control

The range controller is controlling the range to the edge of a path. The input is the range error detected by SonarAnalyzer4.vi and the output is the reference for the steering controller. It replaces the function of the bearing controller in landmark legs by providing flow of data for edge following motion.

When following the edge of a path, the navigation system uses range control in place of bearing control to provide the steering of reference. The aim of range control is to monitor the range from the robot to the edge of the path and to steer the robot to correct for variations. When the path veers to the left (say) the range increases and the robot steers to the left.

As path following is a continuous motion, with considerable lags a PD loop is used (Figure 11.15). The proportional gain corrects for the error in tracking range. While the derivative gain detects the rate of change of tracking error, effectively providing a feed forward stabilisation of the controller.

Figure 11.16 demonstrates the capability of this control system. The robot travels about 45m along a 2.1m wide corridor and then turns 90° to the left (Figure 12.6) and travels of further 7 meters. The solid line shows the robot position (x, y) as calculated by Titan’s crude odometry. The large curve to the right of this line shows how poor the odometry is, due to the fact that the same pulse to distance calibration constant was used for both side of the robot. Given that the robot was tracking 1m from the left wall, and it is 0.7m wide, it was only 400 mm from the right wall. In contrast, the range to the wall of the corridor has a small error (Figure 11.7) except where there are steps in the left wall.
Figure 11.15 Range tracking PD loop is used in edge following mode. The controller takes the Range Difference as an input and generates a reference steering command. Algorithm in Appendix B.6.
Following a corridor wall with a turn using only encoders data

Figure 11.16 Following a 45 meter corridor with left hand turn to new corridor. The robot demonstrated good performance keeping a steady 1 meter range from the wall and gently responding to range changes caused by niches of closed doors. The drift in the plot shows the effect of using the same pulse to distance calibration constant for both sides of the robot. This experiment demonstrates how well the range control works in the presence of poor odometry.
2.5 Range difference during navigation along corridor wall and turning into the next corridor

Distance in meters. To plot - run PlotRobotExperiment35.m

Figure 11.17 shows the range error for the trajectory in Figure 11.16. The small steps in range are due to door cavities. The large steps at 35 and 40 metres are large entrances to toilets. At 43 metres the error goes off scale as the sensor moves past the corner where the corridor turns left. The robot then turns left in search of the wall. The return of the error value to zero, shows that it successfully found and tracked the new wall. The range error of 2.5 metres is issued automatically when the intensity of the edge signature drops below a threshold, forcing the robot to search for the edge.

11.5 Distance controller

As mentioned at the introduction to this chapter, we explored a few approaches to control the distance travelled along a leg. The solution presented here provides Titan with an accurate result (small fraction of a percent of leg length) while not limiting the travelling speed for long legs. The controller calculates the position along the leg from the odometry and generates a velocity command profile which is composed of a constant acceleration stage, a constant speed stage and a deceleration stage.
At the start of the leg, the robot is accelerated at constant acceleration until it reaches the planned maximum velocity. It then travels at that velocity until it is near the end of the leg. It constantly compares the distance it is yet to travel to a distance we call the distance-to-stop. When the two are equal the robot begins to decelerate. The deceleration rate is calculated to bring Titan's velocity to zero at the end of the leg. At each iteration of the control loop, the deceleration rate is recalculated based on the distance to the end of the leg and the planned distance to stop at the current velocity.

The algorithm was simulated in Matlab and implemented in LabVIEW. The core of the algorithm is implemented in a formula node which generates the rule based control law. The only modification required in practice is the addition of a small output aimed to provide a kick-up start when the robot is at the zero position (start of the leg). This boost is needed since at that singular point (zero travelled distance) the command to the velocity controller is too small to get the robot rolling.

11.5.1 Matlab simulation for distance control

The following Figures were generated by the Matlab programs DistanceControl1.m (Figures 11.8a and 11.8b) and DistanceControl2.m. The figures illustrate the implementation of the algorithm by simulating two examples. In the first one the robot travels a leg distance of 2.5 meters and in the other 5 meters with a top speed of 0.5 meter/sec and nominal distance to stop of 1.5 meters. The distance to stop of 1.5 meter and the top velocity of 0.5 meter reflects the predefined constant acceleration. At each increment along the path a suitable reference velocity is calculated. Distance based formulation creates a trapezoidal shape reference velocity with parabolic curved sides. In contrast, a time based representation has straight line sides which means constant acceleration or deceleration (0.083 meter/sec² in the current design). The gradual change in velocity given by the distance based formulation provides a gentle start and stop without oscillations. It is also interesting to note that since the algorithm is a distance dependent and not time dependent, it is not sensitive to transient speeds.
or time delays (for example, during the starting of the motion it takes time for the velocity to ramp up due to the inertia of the robot. The basic dynamics underlying the algorithm is:

\[
\text{ConstantAcceleration} = \frac{(\text{TopVelocity}^2)}{(2\times\text{NominalDistanceToStop})} \tag{11-1}
\]

\[
\text{ReferenceVelocity} = \sqrt{2\times\text{ConstAcceleration}\times\text{CurrentDistanceToStop}} \tag{11-2}
\]

Reference velocity value is clamped to top velocity (x4).

```
1  % The program DistanceControl11.m simulates the LabVIEW program DistanceControl4.vi
2  % which controls the travelling distance of the robot. The leg distance (or new
3  % leg distance, if corrected by recognition during navigation) is stored in the
4  % navigation map. The algorithm generate a trapezoidal shape reference command
5  % which is used as an input to the velocity controller.
6
7  % Constant acceleration motion is based on basics physics Velocity = Acceleration*Time
8  % and TotalledDistance = 0.5*Acceleration*ElapsedTime^2
9  % The "distance to stop" formula issues reference velocity:
10 % ReferenceVelocity = sqrt(2*ConstAcceleration*CurrentDistanceToStop) clamped to x4
11 %
12 % referring to the input/output definitions in the Formula node in DistanceControl4.vi :
13
14  k = 1; % indicator for full developed velocity
15
16  x3 = 2.5; %5; %0.5 %10; % the leg distance in meter
17
18  dx1 = 0.001*x3;
19  x1 = 0:dx1:x3; % the travelled distance in meter of mid-rear axle
20
21  x4 = 0.5;
22 % the velocity clamp value in m/sec (top velocity)
23
24  x5 = 1.5; % distance to stop (the acce/deceleration transient) in meter
25
26  y1 = (x4^2)/(2*x5); % constant acceleration or deceleration
27  yl = 0.083333333333333 (m/sec^2)
```

**Figure 11.18a** Matlab code segment used for the simulations of the reference velocity for distance control (first part from DistanceControl11.m).

In the first example, we simulate a path that is too short (2.5m) for the robot to reach maximum velocity. As a result, the second stage is missing, and the robot has to start decelerating before it has finished accelerating (Figure 11.19). The result of this algorithm is that the distance the robot has moved along the path varies smoothly (Figure 11.20).
When the reference velocity is plotted against distance (Figure 11.19) the curves are parabolic. When the actual (simulated) velocity is plotted against time the sides are straight due to the constant acceleration.

Figures 11.22 to 11.24 show the results of the second simulation, where the path length of 5 metres, results in the robot running at the maximum velocity for a 2 meter stage.

```matlab
% The program divides the leg into two halves and inspects the velocity. % Each half leg can consist of only a transient (for peak velocity smaller than % top velocity x4 in case of a short leg) or transient section plus constant velocity section for % long leg. For short leg there are only two time zones (t1 and t3). For long leg there are four % time zones (t1, t2, t3 and t4).
for i = 1:length(x1)
    if (x1(i) <= 0.5*x3) % travelled distance smaller of equal to half leg length
        y2(i) = (2*y1*x1(i))^0.5; % velocity command as a function of distance
        t1(i) = (2*x1(i)/y1)^0.5; % time to contact = sqrt(2*travelledDist/ConstAcc)
        t2(i) = max(t1) + sum(dt2(1:i));
        k = k + 1; % k>1 indicates long leg with constant velocity section
    end
    if (x1(i) > x4) % constant velocity section
        y2(i) = x4;
        dt2(i) = dx1/x4;
        t2(i) = max(t2) + sum(dt2(1:i));
    end
    else % short leg is without top velocity section
        t2 = zeros(size(t4));
        t3 = zeros(size(t4));
    end
    if k == 1 % short leg is without top velocity section
        t4(i) = 2* max(t1) - (2*(x3-xl(i)/y1)^0.5;
    else
        t4(i) = max(t3) + max(t1) - (2*(x3-xl(i)/y1)^0.5;
    end
end % travelled distance smaller of equal to half leg length % velocity command as a function of distance % time to contact = sqrt(2*travelledDist/ConstAcc) % constant velocity section % elapsed time along the leg
end % travelled distance smaller of equal to half leg length % velocity command as a function of distance % time to contact = sqrt(2*travelledDist/ConstAcc) % constant velocity section % elapsed time along the leg
```

Figure 11.18b Matlab code segment used for the simulations of the reference velocity for distance control (second part from DistanceControll.m ). Algorithm in Appendix B.7.
Figure 11.19 Matlab simulation for the reference velocity command as a function of leg distance for a very short leg of 2.5 meters. The length of the leg in such a case is less than twice the 1.5 meter of distance to stop, thus the velocity does not ramp up to its maximum planned value of 0.5 m/sec.

Figure 11.20 The soft start and stop of the distance as a function of time for a 2.5 meter leg.
Figure 11.21 The behaviour of the velocity along the time axis for a 2.5 meter leg.

Figure 11.22 Reference velocity command as a function of distance for a 5 meter leg.
Figure 11.23 The soft start and stop of the distance as a function of time for a 5 meter leg.

Figure 11.24 The behaviour of the velocity along the time axis for a 5 meter leg.
Figure 11.25 LabVIEW implementation for the distance controller VI. The diagram block of DistanceControl4.vi reads the leg distance (for bridging leg without recognition, or new leg distance for landmark leg with landmark recognition) command from the navigation map. The generated velocity profile also effects the directional clamp for the bearing controller while gradually starting or finishing the travel along the leg. Algorithm in Appendix B.8.
Figure 11.26 Experimental data from a 7 meter leg implemented by the current solution DistanceControl4.vi. The velocity command has a trapezoidal shape with soft start. The velocity feedback is quite noisy since the rear encoders (2500 pulses per one revolution) are driven by friction wheel with 0.49mm/pulse. The large inertia of the robot smooths the actual motion. The noise source is mainly synchronisation problem in the calculation of pulse increments and time increments and does not reflect the physical velocity behaviour.

11.5.2 LabVIEW implementation experience

The original approach was to measure the current velocity and add or subtract an amount based on the desired constant acceleration and the time increment in a do-while loop until the desired top velocity was achieved. The decision if the robot is in acceleration mode or deceleration is done by gauging the distance along the leg and judging at which stage the robot is travelling. This approach yields poor results in controlling the distance because of the noisy velocity signal (Figure 11.26) which is not suitable for calculating fine trimming corrections.

First we examine the velocity VI and changed it to measure the time stamp more accurately. The result is less noise on the velocity signal (Figure 11.27). We believe that with a less noisy measurement of the velocity, the original approach will work.
However, we tried the alternate solution of calculating the reference velocity using the distance-to-stop value. The distance measurement is much less noisy compared to the velocity and therefore suitable for calculating the reference velocity. The *labVIEW* implementation is shown in Figure 11.25. The result is presented in the section (Figure 11.26 and 11.27) were obtained using the implementation. The trajectories in Figure 11.27 were for a 5m leg (top/left) and for a 30 metre leg (top/right and bottom images).

With this control, the robot typically topped within 25 mm of the commanded distance for a 5 to 30 metre leg.

![Figure 11.27 Actual velocity of robot (noisy signal) and velocity reference for 3 runs of robot. The third graph (30 meter leg) shows improvement from reducing the noise on the velocity signal by forcing more synchronised calculation.](image)

11-29
11.6 Turn controller

In Section 11.3, we saw that the bearing controller overshot by a large amount when commanded to change heading by 90°. In this section, we investigate a method of controlling the robot to travel around an arc to achieve the desired change in heading without overshoot. The turn controller and bearing controller are connected in a hybrid control system (Figure 11.3) that switches between bearing control on straight segments and turn control on bearing changes.

When turning a corner a car driver smoothly increases the angle of the steering wheel during the first half of the corner and smoothly decreases it during the second half. The path followed by the car from one straight to another is a complex but smooth curve. It is desirable that a mobile robot follows such a smooth transition through a curve.

However, when the path is specified as a straight segment, a circular arc segment and a straight segment, the robot is physically unable to follow it. An attempt to follow such a path will result in response lags that cause tracking errors. The traditional solution to this problem is to plan a smoothly curving trajectory that enables the robot to stay within the constraints imposed by its dynamics.

Both clothoids and quintic polynomials enable the planning of smoothly changing curve segments between straight path segments. However, clothoids require integration along the paths because they have no closed form solution. A forth order polynomial in polar coordinates meets the path constraints for an arc turn [Nelson, 1989]. A Quintic polynomial meets the constraints for a lane change path. Both types have closed form solutions.

Once the smoothly curving path is planned, then a trajectory that doesn’t violate the robot’s dynamic constraints can be determined. The latter requires a model of the robot’s kinematics [Burke and Durrant-Whyte, 1993] and dynamics [Zhao and BeMent, 1992]. To follow this trajectory in real time, the robot controller must use the dynamic model to calculate the references for its position and velocity control loops at regular intervals. Due to the
difficulty of accurately measuring the dynamics, many mobile robots do not include a
dynamic model in their control systems.

In this section, we present an alternate approach, that is much easier to include in a
control system. Biologists studying birds landing on peaches have found that the trajectory
they follow can be described mathematically. This trajectory decelerates the bird to zero
velocity at the point of contact. It is based on sensing the flow of motion, using either optic or
acoustic sensors, and is called time to contact [Lee, et al., 1992].

When a robot is commanded to move a specified distance along a straight path in
minimum time, it must follow a trapezoidal velocity profile. In Section 11.5, we presented a
distance to stop algorithm that is based on the time to stop concept. This algorithm generates
a trapezoidal velocity profile as the robot travels along the path.

When a robot path is planned as a sequence of straight and arc segments, the tracking
error grows rapidly. Here, we present an angle to bearing algorithm, again based on the time
to stop concept. This algorithm generates an angular velocity profile to enable the robot to
turn through the desired angle, around the desired radius, and correct for tracking error. It can
command a robot to track a circular arc while travelling with either fixed or varying linear
velocity.

Both the distance controller and the turn controller are a control law based on
trapezoidal pattern with parabolic sides (the base of the trapeze is linear distance for distance
control and angular distance for turn control, while the vertical axis is the linear reference
velocity or the angular reference velocity): The distance controller generates the reference
velocity as a function of travelled distance while the turn controller generates the reference for
steering angle as function of the travelled angular distance. The distance to stop is constant
value (1.5 meters) since the purpose of the distance controller is to achieve accurate transient
based on constant acceleration or deceleration. The angle to stop is 1/6 of the angular distance
of turn (so it changes from turn to turn depending on the size of turn) since the purpose of the
turn controller is to achieve 2/3 of the turn with constant steering angle (constant turning radius) while the issue of timing in the transient performance is not important.

11.6.1 Matlab simulation for turn control with constant forward speed

The approach taken is to determine the angle from one heading to the next and the radius of the curve from the map. Again the trajectory is divided into 3 stages: acceleration, following a constant curvature and deceleration. The Matlab simulation program is SmearingTurningRadius2.m

The acceleration stage is always one sixth of the angular distance. During this stage, the steering reference is progressively changed from straight to the reference required for the constant curvature stage. The change follows a parabolic curve. Figure 11.28 shows the parabolic relationship between the resultant angular velocity and angular distance. Figure 11.29 shows the angular velocity versus time.

During the second stage, the robot tracks around an arc with constant angular velocity and constant forward speed. The angular velocity is determined by the steering angle, which is set to give the required turning radius.

Again, the deceleration stage is set to one sixth of the angular distance. Angular deceleration is achieved by progressively reducing the steering angle to 0°, following a parabolic function.

In Figure 11.30, we see that this simulation results in the correct change in heading of the robot but it ends up on a parallel trajectory. During the acceleration stage the radius of curvature is larger than that of the curve and the actual trajectory does not keep up with it.
Figure 11.28 Angular velocity (Rad/sec) command as a function of travelled angular distance (Rad) - based on 0.5 m/sec constant linear velocity assumption, 0.765 m planned radius and 90 degrees turn.

Figure 11.29 Angular velocity (Rad/sec) command as a function of time (sec) along the curve - based on 0.5 m/sec constant linear velocity assumption, 0.765 m planned radius and 90 degree turn.
In the previous section, we simulated a robot transitioning from a straight path into an arc and out again, while moving at a constant forward speed. In the navigation experiments with discontinuous landmarks discussed in Chapter 12, the robot always stops to scan the landmark before entering a curve. The simulation program (Figure 11.32a) for the general case is AngularVelocityTurn3.m. The program is using complex integration (for 3+3 functions describing the radius and x-y component for each transient) the motion shared by
the angular activity (trapezoidal pattern of steering angle versus angular distance) that effects
the size of the momentary turning radius and by the accumulated linear distance which effects
the momentary linear velocity (trapezoidal pattern of reference velocity versus travelled
distance).

In simulation, we accelerate the robot from a stopped position at the start of the curve. So the acceleration stage includes both linear and angular acceleration. In figure 11.31, we see the angular velocity curve has a step in it where the angular acceleration stops while the linear acceleration continues.

Figure 11.32 shows that the simulated robot follows the planned trajectory, but again with a delay (as in Figure 11.30) which results in the robot finishing on a parallel path.
Figure 11.31 Simulation of angular velocity during controlled 90° turn. The simulation reflects the ramping up of the linear velocity by the distance controller based on the same principle of time to contact rule base. The pattern is dependent on complex integration of 6 functions.
Figure 11.32 Simulation of the actual controlled 90° turn versus planned 1.5 meter turning radius. The simulation reflects the ramping up of the linear velocity by the distance controller based on the same principle of time to contact rule base. The pattern of the curve is dependent on complex integration of 6 functions. The straight lines divide the simulated curve into three sections. It is evident that the transients extend more than $1/6$ of the length of the curve because of the larger radius during the transition areas (each transient occupy $1/6$ of the angular distance of the turn).
The program `frigularVelocityTurn3.nl` simulates the pattern of angular velocity during a 90 degree controlled turn. This simulation links the linear motion (which has a velocity profile dependent on the travelled distance) with the pattern of the steering angle (which depends on the travelled angular distance).

The program calls 6 user defined functions for 2 transitions (the first 1/6th and the last 1/6th) at the start and finish of the turn.

```
bel = 90/57.3; % The angular distance of the turn
r1 = 1.5; % Planned turning radius
x4 = 0.5; % Top linear speed
x5 = 1.5; % Distance to stop (from top speed to zero)
a1 = (x4^2)/(2*x5); % The angular distance of the turn
pl = top linear speed
pl = top linear speed
distance to stop (from top speed to zero)
a2 = (x4^2)/(2*x5);

lt2 = 0.001:bel:2*0.001:bel:0.001; % The angular distance of the turn

For i = 1:length(lt2)
if lt2(i) = bel/6
    dsl(i) = quad8('TransitionTurning1',0.001,lt2(i));
    vl(i) = sqrt(2*a1*ds1(i));
    if vl(i) > x4
        vl(i) = x4;
    end
    trl(i) = TransitionTurning1(lt2(i));
    omegal(i) = vl(i)/trl(i);
    dslx(i) = quad8('ComponentX1',0.001,lt2(i));
    dsly(i) = quad8('ComponentY1',0.001,lt2(i));
elseif (lt2(i) > bel/6 & lt2(i) <= bel*5/6)
    dt2 = (bel-2*0.001)*0.001*r1;
    dt2x = (bel-2*0.001)*0.001*r1*cos(lt2(i));
    dt2y = (bel-2*0.001)*0.001*r1*sin(lt2(i));
    ds2 = max(dsl) + sum(dt2(l:i));
    v2(i) = sqrt(2*a1*ds2(i));
    if v2(i) > x4
        v2(i) = x4;
    end
    omegaz = v2(i)/r1;
    dsz = max(dsz) + sum(dt2x(l:i));
    dsy = max(dsy) + sum(dt2y(l:i));
else
    ds3 = max(dsz) + quad8('TransitionTurning2',bel*5/6,lt2(i));
    v3(i) = sqrt(2*a1*ds3(i));
    if v3(i) > x4
        v3(i) = x4;
    end
    tr3 = TransitionTurning2(lt2(i));
    omegaz = v3(i)/r3;
    ds3x = max(dsz) + quad8('ComponentX2',bel*5/6,lt2(i));
    ds3y = max(dsy) + quad8('ComponentY2',bel*5/6,lt2(i));
end
omega3(l:length(omega)) = omegal;
omega3(l:length(omega)) = omegal;
```

Figure 11.32a Matlab code segment for simulation of a controlled turn combined with ramping up linear velocity controlled by distance controller. Algorithm in Appendix B.9.
11.6.3 LabVIEW implementation for the turn controller

The controller was implemented in LabVIEW (Figure 11.33) using a formula node to calculate the steering reference for each angular distance around the curve. A sub-VI (Figure 11.37) calculates a reference steering angle for the curve radius defined in the navigation map.

Figure 11.34 shows the results of the robot following such a curve, starting from a standstill. The results are similar to the simulation results. The turn controller is included by a dynamic call with the bearing controller to form a hybrid direction controller (Figure 11.3).
Figure 11.34 Using LabVIEW for turn control experiment shows recording of the actual turn compared to planned turn for a 90° and 0.765 meter radius. The actual radius is smeared by 0.2 meter due to the effect of the parabolic transition of the curvature following the trapezoidal pattern.

Figure 11.35 shows the results of driving the robot along a 6 leg rectangular spiral trajectory. Comparing Figure 11.35 to Figure 11.13, we see that the overshoot on 90° turns has been removed. This is confirmed by comparing the actual heading of the robot in both cases. (Figure 11.36 to Figure 11.14)
Figure 11.35 Robot’s trajectory in comparison to planned trajectory. The small circles represent the centres of the turning radii. Applying the feed-forward ruled based turn controller eliminates the bearing overshoot. Better tuning of the bearing controller could help to reduce drifts along the leg.
Figure 11.36 Compass response to bearing commands shows reduced overshoot due to the feed-forward action of the turn controller.
11.6.4 Steering angle calculation

The navigation map specifies a turn radius. The VI in Figure 11.37 calculates the steering angle for both left and right turns using the steering kinematics model (Figure 11.4). Left and right turns are different because the steering encoder measures the angle of the left wheel. The variations in steering angle during two turns with different radii is shown in Figures 11.38 and 11.39.

![Diagram of RadiusSizeControl.vi](image)

**Figure 11.37** The radius size controller converts the specified turning radius to steering reference angle which depends also on a left or a right turn. This VI is embedded as a sub-VI in the TurnControl3.vi.
Figure 11.38 turning right 90° with a 0.765 meter radius combined with a ramping-up velocity.

Figure 11.39 Turning right 90° with a 2.5 meter radius. The reference steering command has a symmetrical trapezoidal shape (as a function of turn angle) composed of a 15° rise followed by a 60° flat and resuming by a 15° descend. The linear velocity is ramping up during the turn which explains why the slopes of the time dependent trapezoidal are different.
12 Navigation experiments and results

In this chapter, we describe several navigation experiments and discuss their results. The goal of this thesis was to develop an outdoor robot that could navigate using CTFM ultrasonic sensors to detect landmarks. The chosen navigation strategies are based on those commonly used when piloting light aircraft.

In chapter 2, we first described the navigation and sensing strategies. In subsequent chapters, we described the components of the system in detail, including their design, development and testing.

This chapter completes the thesis by examining the results of several experiments. These outdoor navigation experiments were conducted in 3 venues:

1. In the university campus, mainly on the large cricket field across the road from the school of IT & CS. This field provides a huge space without obstacles which is important for testing a large mobile robot. It would be very difficult to conduct multi-leg navigation experiments in cluttered environments, where there is a high risk of collision.

2. At the ANU farm near Canberra (Australian Centre for Field Robotics) during 5 overnight trips. This unique place enables us to refine the motion control of the robot while navigating between different types of crops. Also, it highlights the robot’s susceptible to back-driven effects from the farm’s bumpy ground. Another interesting challenge exposed at the farm was the coupling (from the aspect of CPU resources) between sensing and motion. The logistic aspects of transporting the robot 250 km to the testing field was challenging and included building special ramps for loading and unloading the robot. Interestingly, the navigation algorithm (including velocity and distance control) for following a path edge was later used to demonstrate autonomous climbing on this ramp.

3. Wollongong Botanic Garden is next door to the university campus. It contains many long paths that we used to test the robot for long (250 meter single leg) path following, including coping with junctions.
The indoor navigation experiments were conducted in the long (45 meter) corridor in our building. We used the corridor to explore multi-path sensing problems, to navigate along a corridor and to learn how to round a corner into the next corridor. This experience helped us later in following junctions in the botanic gardens.

12.1 Recording experimental data

Data was collected every 250 msec during an experiment and stored in a file for later analysis. The 26 global variables are grouped into four categories: Navigation data, Motion data, Control data and Recognition data. Post processing was done with Matlab. The labVIEW program DataRecorder3.vi for data recording is shown in Figure 12.1.
Figure 12.1 LabVIEW implementation for the data recording of 26 variables. The recording enables the researcher to perform post processing such as plotting the robot's trajectory or analysing any problematic issues that occurred during the experiment. Algorithm in appendix B.11.
12.2 Navigation between isolated landmarks without recognition

To test the ability of the robot to navigate using odometry and compasses only, we conducted experiments to navigate between landmarks without localisation. These experiments give a measure of the open loop navigation ability of the robot, as a reference for comparisons to landmark navigation. Also, this type of navigation is equivalent to flying blind (in bad visibility) using instruments only.

In the first experiment, the robot was commanded to follow a path consisting of 6 legs (Figure 12.2). These legs are called bridging legs because they usually occur as legs between landmark navigation legs, when suitable landmarks are not available.

The ability of the robot to follow the path in Figure 12.2 is very good. Errors in the path following only occur due to the lag on turns (Section 11.6). As the path includes both left and right turns in a regular order, errors on subsequent turns compensate rather than accumulate.

The bearing control (Section 11.3) achieves very good straight line travel on a smooth open field. The ability of the hybrid controllers to control steering is shown in Figure 12.3. The compass reading, and bearing error, show clean steps on 90° turns without overshoot.

The robot performs a sequence of bridging legs using compass and odometry (Figure 12.2). The motion control is identical to navigation with recognition of isolated landmarks. Each leg consists of a combination of small radius (0.765m) curved section and a straight line section. The combination depends on the navigation map. Many experiments were conducted to refine the performance of the motion control and the navigation methodology. Figures 12.2 to 12.5 are typical results.
Figure 12.2 Six leg navigation with a constant turning radius performed on the university cricket field. The actual trajectory is depicted on the background of the planned trajectory generated by the Matlab simulation. The start of each leg is shown with the centre of the turning radius.
Figure 12.3 Bearing error (lower curve) and compass reading in degrees for the navigation in previous figure.

In the second experiment, the path included curves with different radii (Figure 12.4). Again errors of up to 0.4 metres lateral translation are due to lags on turns. Again, the turn controller drove the robot around each curve until the bearing controller took over and reduced the bearing error to zero (Figure 12.5).
Figure 12.4 Five leg navigation with a variable turning radius performed on the university cricket field. The actual trajectory is depicted on the background of the planned trajectory generated by the Matlab simulation. The start of each planned leg is shown with the centre of the turning radius.
12.3 Navigation along continuous landmarks

To test the ability of the robot to follow a continuous landmark using measurement of lateral range to the landmark only, we conducted experiments to navigate along an edge. These experiments give a measure of the closed loop navigation ability of the robot. This type of navigation is equivalent to a pilot usually following a coast line or a river.

In edge following mode, the robot follows a path edge or any continuous landmark by identifying the edge and using the flow of information to feed the motion control without using the compass. The compass was used only to record the trajectory. The first experiments were conducted along a long corridor (Figures 12.6 to 12.9) with a left hand turn to a second corridor. Other experiments were conducted on paths in the botanic gardens.
12.3.1 Edge following of long corridor and turn to new corridor

Figure 12.6 Plan of corridor. In experiments, Titan started at the bottom and travelled along the corridor, finishing with a left turn at the far end. Note, large recesses for doors into toilets at far end.
Figure 12.7 Titan in corridor in Figure 12.6. Left turn is at far end.
Figure 12.8 Following a 45 meter corridor with left hand turn to new corridor. The robot demonstrated good performance keeping a steady 1 meter range from the wall and gently responding to areas such as niches of closed doors. Refining the odometry calibration avoided the drift in the plot shown in Figure 11.16. It is interesting to compare with Figure 12.9 based on fused data of odometry and ineffective compass.

In the first experiment, the robot was placed in an indoor corridor (Figures 12.6 and 12.7), and commanded to travel a distance of 55m following the left wall. The odometry calculation of location (X,Y) was refined by calibrating the rear wheels encoders, thus producing a straight line trajectory (Figure 12.8) instead of the curvy trajectory (Figure 11.16).

In contrast, the calculation of location (X,Y) based on odometry calculations of distances moved fused with compass reading of direction is usually much better (as can be clearly seen from the results in the botanic gardens in Figures 12.11a and 12.11b) when the
robot is outside. However inside, it is not good due to the poor performance of the compass inside a steel reinforced concrete building.

However, as shown in 12.10, the range error is constrained to be sufficiently small that the robot follows the corridor and turns into the next corridor without colliding with either wall.

![Graph showing robot trajectory](image)

**Figure 12.9** The same trajectory as Figure 12.8 but fused with the compass data. The compass (used only for post processing) is not effective in the building because of the steel framing in the walls and steel reinforcing in the floors.
2.5 Range difference during navigation along corridor wall and turning into the next corridor

Figure 12.10 shows the range error for the trajectory in Figure 12.6. The small steps in range are due to door cavities. The large steps at 35 and 40 metres are large entrances to toilets. At 43 metres the error goes off scale as the sensor moves past the corner where the corridor turns left. The robot then turns left in search of the wall. The return of the error value to zero, shows that it successfully found and tracked the new wall. The range error of 2.5 metres is issued automatically when intensity of the edge signature drops below threshold, forcing the robot to search for the edge by turning left.

12.3.2 Edge following in Wollongong Botanic Gardens

The second experiment, was to navigate along 250 metres of concrete path in the Botanic Gardens (Figures 12.11 a and b). The path had numerous curves in it, including two forks (Figures 12.12 and 12.13) and a T-intersection (Figure 12.14). For most of its length the path (and forking paths) were bordered by grass or gardens. However, opposite the T-intersection the grass was replaced by a very rough concrete path (Figure 12.14) on the left side of the robot.
Figure 12.11a Navigation of one leg with a length of 250 meters in Wollongong Botanic Gardens using edge detection of concrete path flanked by grass. The recording of the trajectory was using only the odometry data. Comparing to the more accurate pattern 12.11b (backed by the precision of the stabilised compass), the result is quite good in the sense that it retains the main features of the navigation.
Figure 12.11b Navigation of one leg with a length of 250 meters in Wollongong Botanic Gardens using edge detection of concrete path flanked by grass. The recording of the trajectory was using the odometry data fused with the compass data, but the navigation did not use the compass and rather a flow of sensing. Comparing version b (which is the realistic
situation) with version a, we can notice that version a retains the main features of the trajectory but the whole pattern lacks angular correctness, which is quite understandable.

Figure 12.11b shows a plot of the path followed by the robot (distance-heading→x-y plane) calculated with odometry (distance) and heading (compass). The changes in general bearing of 90° occur at the forks where the path divides. Figure 12.15 shows the range error as the robot from the left edge. At the curves, at the two junctions, a larger tracking error occurs due to the feedback nature of the control system.

Figure 12.12 Titan navigates autonomously along a path edge. The figure depicts coping with the first junction by concurrently detecting the edge in the signature (represented by the first peak above threshold) and the absolute level of power in the signal. If power drops below certain level, the robot understands that the edge was lost and it invokes a predefined search for the edge.
Figure 12.13 Climbing up-hill to cope with the second junction.

Figure 12.14 Titan handling the third junction. It skipped the rough path and proceeded forward on the concrete path since it could detect a clear edge between the concrete and the dirt path.
The robustness of the system was increased by monitoring the intensity of the echo from the path edge. If the intensity fell below a threshold, the robot was considered to have lost the edge. The recovery strategy was to steer to the left (as in the corridor example) looking for the edge. The edge was re-established when the intensity increased above the threshold.

### 12.4 Navigation with recognition of isolated landmarks

To test the ability of the robot to localise using landmarks we conducted experiments to navigate between poles on a field. We used the visual flight rules approach (Section 2.3) as a strategy for landmark navigation with poles as simple-discontinuous landmarks.
A cylindrical pole is a simple discontinuous landmark (Chapter 8). Common objects in an outdoor environment including fence posts, light poles and tree trunks are members of this class. The navigation experiments with a sequence of poles as landmarks, discussed in the following sections, demonstrate the robustness of navigation with this type of landmark.

A cylindrical pole is symmetrical, so its echo varies little with rotation. As a result it can be detected from almost any angle, with a single feature set. The advantage is that it gives accurate and robust bearing and range readings relative to the sensor. However, to get the bearing requires panning across the pole.

We control sensor motion by stopping the robot and panning the sensor (Section 8.4). The result is a fixed range reading while the pole is in the region of insonification (Figure 8.2). The range value is fixed because of the smooth surface results in specular reflection, and consequently the echo is from the surface element that is orthogonal to the beam axis.

The measurements reported here were made with a CTFM phased array (Chapter 4) mounted on the front left corner of the robot. This sensor produces an elliptical beam of ultrasonic energy with a horizontal beam angle of 3.5° (axis to side of beam) and a vertical beam angle of 30°.

When the pole is not in the region of insonification, a small amplitude echo will be received from the pole. If the pole is isolated from other objects, it will continue to be the largest amplitude echo. In which case, we have to threshold the echo to isolate the reflection from the main lobe (Figure 8.5), in order to determine the bearing of the pole (Figure 8.1). As the pole is symmetrical, the vector along which the range is measured passes through the centre of the pole. The location of the centre can be calculated using simple geometry.

12.4.1 Teaching a path

The robot is taught a path by manually driving it along the path. At each sensing point, it is stopped. The distance and bearing from the previous path or landmark point is calculated and stored in the map. The ultrasonic sensor is panned across the landmark and a signature is
calculated for use in recognizing the landmark. The ultrasonic sensor is on the front left of the robot, so all landmarks used in these experiments were to the left of the robot.

For a symmetric landmark, such as a pole, the signature is the acoustic density profile. For a more complex asymmetrical landmark, such as a bush, the signature is a set of features that are extracted from the acoustic density profile.

The robot navigates the path by traversing each leg in the sequence. It stops at each landmark point and scans for the landmark. Once it has located the landmark, it can determine the error in its location relative to the planned path.

In these experiments, the 22.5 mm poles were detected by taking 51 readings in 1° steps over 50° centred on the expected bearing of the landmark (section 8.4). To find the bearing to the pole, these readings are cross correlated with features calculated from the reference acoustic density profile (Figure 8.1).
Figure 12.16 Five leg navigation demonstrating the capability of Titan to correct the initial localisation error. The landmarks are wooden poles stuck in the grass. The figure depicts the actual trajectory (green line which starts with initial deviation from the planned position) and also the planned trajectory (blue line - colours could be seen if using the Word file of the attached CD or running the Matlab simulation in the CD). The planned position of the mid-rear axle of the robot at each landmark is marked by a small circle linked with a short line to the planned trajectory. It is also representing (the centre of the small circle) the theoretical centre of the turn. The navigation algorithm shows good convergence.
12.4.2 Results

To test the visual flight rules strategy, we set up paths across a sports field using poles as landmarks (Figure 12.16). The robot has successfully navigated these paths on several occasions. Comparing Figure 12.16 to Figure 12.2 and 12.4 shows that the use of the isolated landmarks for localisation significantly reduces the errors in path tracking. One such transversal of the path in Figure 12.16 path is given in Table 12.1.

Columns 1 and 2 show the planned distance and compass bearing for each leg. Columns 3 and 4 shows the difference in the range and bearing of the landmark when measured by the sensor to the values in the planned map. Columns 5 and 6 show the new leg parameters for the leg, calculated from these differences.

<table>
<thead>
<tr>
<th>Leg</th>
<th>Leg length</th>
<th>Leg angle</th>
<th>Range error</th>
<th>Angle error</th>
<th>New leg length</th>
<th>New leg angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17m</td>
<td>45°</td>
<td>0.35m</td>
<td>-6.7°</td>
<td>16.96m</td>
<td>44.12°</td>
</tr>
<tr>
<td>2</td>
<td>10m</td>
<td>0°</td>
<td>-0.06m</td>
<td>-6.42°</td>
<td>9.965m</td>
<td>0.296°</td>
</tr>
<tr>
<td>3</td>
<td>25m</td>
<td>75°</td>
<td>-0.04m</td>
<td>-6.9°</td>
<td>25.04m</td>
<td>75.02°</td>
</tr>
<tr>
<td>4</td>
<td>30m</td>
<td>150°</td>
<td>0.000</td>
<td>1.38°</td>
<td>30.0m</td>
<td>150°</td>
</tr>
</tbody>
</table>

Table 12.1 Map, errors and re-planned segments for the path in Figure 12.15.

To produce the data in Table 12.1, the robot was deliberately placed in an incorrect start location relative to the 1st pole. It was 350 mm further from the landmark and the compass bearing from the sensor to the pole (angle error in Table 1) was 6.7° to the previous sensing bearing. To correct for this error, the first leg was reduced in length by 40mm and the bearing changed by 0.88°. These changes reduced the range error at the next landmark to 60 mm and the error in the sensing vector angle to 6.42°.

The data in the table shows that even over relatively large distances, the errors are small. However, these errors can accumulate from one leg to the next and become quite large if not
corrected. The corrections serve to reduce the error as the robot travels toward the next landmark, as shown in Figure 12.17 for another experiment.

Figure 12.17 Tracks on grass show the convergence of a re-planned straight leg with the planned leg toward the end point of the leg.
Figure 12.18 Two leg navigation which includes three different experiments. The first one (1) was navigation without recognition according to map (left plot). The second one (2) was again navigation without recognition but with introduced angular bearing in the first leg of the map, causing the second leg to develop parallel deviation (right plot). In the third (3) experiment the robot used recognition of the second landmark to correct the navigation and arrive to the third landmark in correct position (middle plot).

12.5 Conclusion

Figure 12.18 compares the robot following a two leg path with and without landmark sensing. First the recognition and navigation map were recorded (left graph). Second, the Titan was navigated using odometry and compass only with initial angular error introduced to the first leg map. The result is the right graph showing that an initial angular error is not corrected.

In contrast, with the same initial angular error, the landmark navigation changed the second leg to correct for this error, and the robot is much closer to the desired location at the end of the second leg (middle graph).

The results in this chapter demonstrate that we have achieved the goals of the thesis. We have-

A) Designed and built robot for outdoor landmark navigation.

B) Developed sensing strategies for robustly detecting both isolated and continuous landmarks

C) Designed and implemented algorithms for landmark navigation based on the Visual Flight rules (VFR), Instrument Flight Rules (IFR) and coastline navigation strategies used by small aeroplane pilots and

D) Demonstrated that a complex mobile robot control system can be developed by first simulating (in Matlab) and then implementing (in labVIEW).
13 Conclusions and future research

In this thesis, we set out to navigate the Titan robot along a path made up of a sequence of legs using landmarks detection with a CTFM ultrasonic sensor (section 1.9 – page 1.23). Along the way we achieved the following goals:

A. Designed and built a robot for outdoor landmark navigation,
B. Developed sensing strategies for robustly detecting both isolated and continuous landmarks with a CTFM ultrasonic sensor,
C. Designed and implemented algorithms for landmark navigation based on the Visual Flight rules (VFR), Instrument Flight Rules (IFR) and coastline navigation strategies used by small aeroplane pilots, and
D. Demonstrated that a complex mobile robot control system can be developed first by exploring simulations for the design phase (Matlab, Simulink and AutoCAD) and later implemented in real-time environment for testing the product (LabVIEW).

13.1 The contribution of this thesis

The following contribution to mobile robot research were made in this thesis:

A. Proved the feasibility of using the navigation methods used by people when flying light aircraft for autonomous navigation of outdoors mobile robots.
B. Proved the feasibility of using a CTFM narrow beam phase array sonar to recognize outdoor landmarks cluttered by natural environment.
C. Proved the feasibility of a modular design concept which allows great flexibility in upgrading the computer hardware and the programming software environment in contrast to the traditional approach of building a custom built computers and data acquisition interfaces.
D. Proved the feasibility of using engineering software tools with reliable technical support over the Internet by the manufacturers of the tools (reflecting a strong commercial commitment) to advancing mobile robot research.
13.2 Future research work

In this thesis, we developed a research vehicle that can be used in future research, and provided a basis for solving additional problems in landmark navigation. Some of the possibilities for future research are described below.

A. Explore the feasibility of building path edge classes for autonomous navigation in the Wollongong Botanic Gardens and in the University of Wollongong campus. The idea is to extend the navigation map and the recognition map so that the robot will be able to understand and derive conclusions during navigation in uncharted outdoor spaces. This involves developing a model of path edges using features extracted from the CTFM echoes. Then a path edge map can be designed for navigation along paths.

B. Explore the feasibility to use a fuzzy logic controller that will receive as input the data in the navigation map and the recognition map. The controller will also be able to combine data from the inclinometer and the real-time vision. Fuzzy rules could be used for object recognition, for localising the robot on the map and for describing the next path leg.

C. Explore the feasibility of using FireWire (IEEE 1394) or USB2 as an a single standard interface between the laptop and the robotics platform. The underlying idea is to develop a fast micro-controller which will assemble all the motion control parameters (each variable will be presented by 12 bits, exactly as in our NI interface) with other sensory data like the CTFM phase array sonar and real-time vision digital frames (for example – 640x480 pixel with colour depth of 24 bits). The combined data will flow in both directions via the FireWire (on a Mac laptop) or a USB2 (on a PC). At the laptop, a driver will need to separate the vision data from motion control or other data and process it to achieve visual (or any other method) servoing.

D. Based on the design of Titan, develop the concept of a mini Titan (Figure 13.1), using the interface described in C, to provide a low cost robot for education. This feasibility study includes the conceptual design of the Mini-Titan mobile robot in Figure 13.1 . A first step
towards such an educational robot has recently came onto the market (Figure 13.2). The advantages of such an approach include:

- A low cost data acquisition interface which is fast enough for most educational projects in the field of mobile robots.

- A very flexible robot that can be reconfigured for different experiments.

- A low cost modular design suitable for robotics education in colleges and universities.

E. Explore the feasibility for outdoors landmark navigation strategies for use by aerial robots by fusing acoustic flow information and visual servoing data. We plan to test few concepts of hovering robots and hybrid robots and apply this research to the motion control and navigation scheme.

F. Explore the feasibility to build a generic programmable interface for radio controlled (RC) robotic kits using LabVIEW virtual instruments (graphical programming) and USB2 drivers. This could link Mac and PC (desktop computers for indoor training and laptop computers for outdoor training) the huge industry of RC toys to programming training in high schools.
Mini-Titan is a scaled down concept based on Titan's experience and modular design approach.

- Each analog input is represented by 12 bits digital variable.
- 24/32 bit color depth with sensory data in digital format.
- Micro-controller assemblies each vision frame (640x480 pixels with
  computer interface is based on single USB2 channel (wire or wireless)
  disengaged to convert to floating steering system).
- Electric powered Ackerman steering system (mechanically)
- Total weight: maximum 10 kg.

USB2 Robotics Interface
The Evolution Robotics ER1 is the first personal robot system you can easily train to do useful and fun things around the house. All you need is a Windows laptop and your imagination. With the easy-to-use software, you can train your robot to do things like recognize objects and places, send email, take pictures and video, respond to voice commands, and more!

• Not a toy, but a real autonomous robot
• Have it up and running in about an hour
• Train it to help you and entertain you

that's simple to use...
• USB compatible, plug-n-play electronics
• Point and click: command interface

and fun to own!
• Play your favorite music just by holding up a CD
• Get important reminders—wherever you are
• And with the help of the robot arm (future accessory), recognize and grab small items and bring them to you

Laptop not included

Figure 13.2 - The Laptop robot from Evolution Robotics In Pasedana CA USA is sharing a very similar conceptual design of our MiniTitan.
14 Bibliography


Downer, J., 1988, 'Supersenses - perception in the animal world', published by BBC Books, printed in Great Britain by Butler & Taner Ltd. London


Harper, N.L. and McKerrow, P.J., 1999b, Detecting plants for landmarks with ultrasonic sensing', Proceedings FSR'99, Pittsburgh, August, pp 144-149.


Kanayama, Y. and Miyake, N. Trajectory Generation for Mobile Robots, Chapter 8, pp 333-340.


14-5


Appendix A

Content of the CD:

1. Matlab 5.2 programs - Macintosh
2. LabVIEW 5.1 programs - Macintosh
3. AutoCAD LT programs - windows
4. Quicktime movies - Macintosh
5. Thesis chapters Word 98
Appendix B - pseudo-code algorithms

1 Algorithm for Figure 6.23

This algorithm is used by the researcher while operating the Titan robot to add - by a manual process - a single CTFM reference signature to the recognition map. The algorithm reads 15 consecutive signatures from the same landmark at the same pose of the robot with respect to the landmark and creates a recorded file with a CTFM reference signature based on the averaged values of the elements from the 15 signatures.

Landmark signature algorithm
1. Aim sensor at landmark
2. Start a while loop (step 3 → 11)
3. Repeat 15 times using a for loop (step 4 → 8)
4. Read 256 value from echo
5. subtract mean from echoes values to remove dc offset
6. Estimate spectrum (acoustic density profile) with Welch method estimator
7. Save in array of 128 elements (positive half)
8. Calculate running mean. With array of 15 profiles (each compose of 128 elements) calculate the average of the 15 signatures to get one robust CTFM reference signature.
9. Find the position for the signature's peak amplitude
10. Record the reference signature in a data file. The recognition map consists of a set of such data files that represent reference signature for each of the isolated landmark.

2 Algorithm for Figures 10.26 and 10.30

This algorithm is the master navigation program and is used by the Titan robot to navigate autonomously between outdoor landmarks (and also in a limited way in indoor environment). The robot corrects the navigation map by recognising the landmark and assessing the localisation error (done by comparing the best matched measured CTFM signature with the pre-recorded reference CTFM signature from the recognition map). The three strategies employed by this program are:

1. Navigation without landmark recognition. The robot uses rear wheels odometry to gauge travelled distance and on board gyro stabilised compass to gauge magnetic bearing.
2. Same as (1.) but using the CTFM narrow beam sonar to recognize continuous landmarks. The correction of the navigation heading is done by analysing the range data to the path edge instead of using the compass readings (as in strategy 1). The sensing is done while the robot is moving at a constant speed and is triggered by distance increments.

3. The same as (1.) but using the sonar to recognize isolated landmarks and using the error assessment to correct the next navigation leg on the fly. This strategy employed two different sensing modes:

   1. Angular scanning while posing in front of the landmark
   2. Linear scanning while passing the landmark in slow motion

**Combined navigation strategies algorithm**

1. Reset the two analog output channels to zero to set the velocity command to zero. (two voltages command the control of the motors (in the original wheelchair these velocity commands are generated by the joy stick).

2. Reset the global variable LandmarkIndex to 1 (the navigation map starts with landmark 1) and global variable DataRecorder (which invokes the DataRecorded3.vi) to F (Fault).

3. Reset the pan and tilt unit (which carries the CTFM sonar head) to 90 degrees pan and zero degrees tilt.

4. Start the navigation map VI (LegMap6.vi) and the data recorder VI (DataRecorder3.vi) by using dynamic calls to reference nodes.

5. Read from the navigation map the value of the global variable for the specific motion strategy (MotionStrategy). Each mode is manipulated by the global variable Landmark Index dictated by the navigation map. The navigation map provides a choice of 4 motion strategy options designated by the numbers 0, 1, 2 and 3. Option 0 $\rightarrow$ motion without recognition (robot using odometry and compass only). Option 1 $\rightarrow$ motion with recognition of an isolated landmark. Angular sweep is used to assess the navigation error and correct the navigation for the next leg. Option 2 $\rightarrow$ motion with recognition of an isolated landmark. Linear sweep is used to assess the navigation error and correct the navigation for the next leg.). Option 3 $\rightarrow$ Edge following motion. Heading correction is achieved by continuously analysing range data to the path edge.

6. Execute motion strategy 0: a) Read the tilt value for this leg from the navigation map (via a global variable). Move sonar head to the specified tilt angle. b) Use a for loop to invoke a sequence of dynamic calls (OpenVI Reference by Invoke Node) for the 8
relevant motion parameter VIs: encoder left and right, encoder velocity left and right, read compass, bearing controller, steering controller and velocity controller. c) Invoke the distance controller by a dynamic call. The controller quits after the robot travels the desired leg distance. d) Use a for loop to initiate a sequence of dynamic calls which invoke the object reference of the 8 nodes prompting the close all the 8 motion control VIs.

7. Or execute motion strategy 1: a) Read the tilt value for this leg from the navigation map (via a global variable). Move sonar head to the specified tilt angle. b) Reset to zero the global variables NewLegDistance and NewLegBearing. c) Initiate a dynamic call to start the SonarSignature4.vi. The VI is running continuously, reads the CTFM sonar signal and exports the Welch method power estimation of the landmark signature via a global variable. It also includes a sub-vi which compares the current measured signature with the reference signature in the recognition map and produces an output of range difference estimation based on cross and auto correlation techniques. d) Initiate a dynamic call to start the AngularSweepSonar1.vi. The VI conducts an angular sweep of 51 observations. It builds in a buffer a set of extracted features for each of these 51 observations. e) At the end of the sweep the sub-vi RemoveDummySignature3.vi is called. This VI resets to zero all pattern similarities which occurs when signal energy is below a certain threshold. f) The robot compares the collected data with the reference CTFM signature provided by the recognition map and finds the best match between the reference signature from the recognition map to one of the 51-measured signatures. It pans the sonar head towards that observation point. g) The data from this new observation is used now by another sub-vi called CalculatePosition3.vi to calculate the corrected navigation for the next leg distance and bearing by triangulations. h) After panning of the head to the best match, initiate a dynamic call to quit SonarSignature4.vi. i) Initiate a sequence of dynamic calls - by using a for loop - to start 8 relevant motion control VIs (encoders left and right, encoder velocity left and right, read compass, bearing controller) that use the new navigation parameters of the next leg, steering controller and velocity controller. j) Initiate a dynamic call to start the distance controller (DistanceControl8.vi) to drive the robot along the new leg. k) After the robot has travelled the desired leg it initiates a dynamic call of the reference nodes automated by a for loop to quit all the mentioned 8 motion control VIs.

8. Or execute motion strategy 2: exactly like in the previous section 7 but using linear sweep instead of angular sweep (7d).

9. Or execute motion strategy 3: a) Read the tilt value for this leg from the navigation map (via a global variable TiltPosition). b) Move the sonar head to the specified tilt
angle c) Initiate a sequence of dynamic calls to start 9 relevant motion control and sensing VIs (encoder right and left, encoder velocity right and left, sonar capture VI, range control VI, steering controller VI, velocity controller VI and compass reading VI). The range controller VI (RangeControl1.vi) uses the range difference (to the path edge) as an input to a PID controller. The output from this controller is then fed into the steering controller. The bearing controller is not used in this mode but the compass reading is recorded for post-processing analysis but does not contribute to the motion control. d) SonarCapture4.vi VI invokes the capturing of a single CTFM signature of the path edge, triggered by distance increment along the path. The captured signature is analysed by a dynamic call to SonarAnalyzer4.vi which in turn calls a sub-vi (WelchMethodSpectrumEstimator.vi) to estimate the spectrum of the CTFM signature. Another sub-vi (FindFirstPeak3.vi) locates the first significant peak above threshold in the estimated spectrum. The location of the peak is translated into range difference data. e) The range difference data is fed into the range controller. f) Invoke the distance controller (DistanceControl4.vi) by a dynamic call. This VI gauges the travelled distance and stops the robot by issuing a zero velocity command (done gradually, not with a step) when the travelled distance is equal to the leg distance dictated by the navigation map. g) Abort all the 9 motion controllers VIs by using dynamic calls in a for loop.

10. For each of the 4 modes described in paragraphs 6 to 9 - repeat the activity by a for loop according to the number of legs in the navigation map to be executed. The current leg number is reflected by the value of the global variable LandmarkIndex.

11. Abort the navigation map LegsMap6.vi by a dynamic call.

12. Stop the recording session of the data recorder DataRecorder3.vi.

13. Run the initialisation VI to reset all the buffers and I/O registers.

3 Algorithm for Figure 11.10

Steering control algorithm

1. Initiate to zero the steering angle and the history of the waveform chart.
2. Reset to zero the two analog output channels, and set port configuration for the digital IO line.
3. Start a while loop (step 4 \(\rightarrow\) 9)
4. Read the steering angle by combining 8 bit from one port and a single bit from another port.
5. Assemble the 8+1 bits to unsigned 16 bit (0\rightarrow65535) using the sub-vi Better9.vi.
6. Calculate the steering angle using the output from Better9.vi and taking into consideration only each second pulse (so 360 degrees are equivalent to 0.5*2500 pulses).
7. Take the difference between the reference command and the actual steering angle (the steering error) and use it in a formula node to produce a clamped proportional and integral (PI) controller.
8. Send the output of the PI steering controller to a digital to analog channel via "AU Update Channel.vi".
9. Send the output of the PI velocity controller (global variable "VelocityInput") to the formula node and forward the output (y1 = x1) to another digital to analog channel via "AU Update Channel.vi".

4 Algorithm for Figure 11.11

Velocity controller algorithm
1. Reset to zero (outside of the while loop) the history data for Waveform chart "Velocity reading" and the history data for the indicator "PID output".
2. Reset to zero the value of the global variable "VelocityInput" which is used to relay
3. Start a while loop (step 4 \rightarrow 8)
4. the output of the PI velocity controller to the Analog output VI in previous figure.
5. Calculate mean linear velocity (at the mid-rear axle of the robot) by summing the global variables "VelocityRight" and "VelocityLeft" and dividing the result by 2.
6. Subtract the mean linear velocity from the reference velocity command coming from the distance controller using the global variable "Reference velocity".
7. Feed the velocity error to a formula node that calculates the proportional and integration components.
8. Output the result to a global variable "VelocityInput" which is sent to the analog output channel.

5 Algorithm for Figure 11.12

Bearing controller algorithm
1. Reset to zero (outside of the while loop) the history data of the Waveform Chart "Compass reading" which fuses the two global variables: "LegBearing" for the planned bearing in the navigation map and "CompassRead" exported from the compass VI.
2. Reset to zero the history data of the array "PID output" outside the while loop.
3. Start a while loop (step 4 → 8)
4. Subtract compass reading from the leg bearing to produce the directional error.
5. Inject the directional error to a formula node to transform the error in degrees to be in the range of plus and minus 180.
6. If the absolute value of the directional error is greater than 15 (degrees) - invoke the turn controller TurnControl3.vi.
7. Inject the direction error into a formula node that calculates the proportional and integral components.
8. Send the result of the PI controller to the steering controller via the global variable "SteeringInput".

6 Algorithm for Figure 11.15

Range controller algorithm

1. Reset to zero the history data of the array "PID output" outside the while loop.
2. Start a while loop (step 3 → 6)
3. Read the global variable "RangeDiffereance (meter)".
4. Feed the range difference data into a formula node that calculates the proportional and derivative components of the rage controller.
5. Inject the result of the PD controller to a global variable "SteeringInput" (which relates the data to the steering controller).

7 Algorithm for Figure 11.18b

Distance controller simulation algorithm

1. Divide the length of the leg by two.
2. Set the desired linear acceleration to a fixed value.
3. Start a for loop (step 4 → 9)
4. Starting at zero position, increment the travelled distance each time by 0.001 of leg's length.
5. For each position along the first half leg, calculate the theoretical velocity for the desired constant acceleration.
6. Compare the theoretical velocity to the maximum targeted linear velocity.
7. If the theoretical velocity at the momentary position along the half leg is under the maximum speed permitted to the robot, proceed with increasing velocity according the parabolic analytical formula for linear motion with a constant acceleration.
8. If the theoretical velocity exceeds the maximum permitted to the robot, freeze it to the value of that maximum.

9. If travelled distance exceeds half the leg's distance, proceed the motion with a mirrored pattern.

8 Algorithm for Figure 11.25

Distance controller algorithm

1. Reset to zero the history data of Waveform Chart "Forward velocity".
2. Set the digital line logic state to give a computer control.
3. Start a while loop (step 4 → 13)
4. Read the global variables "LegDistance" and "LegBearing" from the navigation map.
5. Read the global variable "LandmarkIndex" from the navigation map.
6. Calculate the travelled distance by summing together (and division by two) the rear encoders odometry.
7. Read the maximum desired top speed of the robot from the global variable "VelocityClamp".
8. Read the desired "Distance-to-Stop".
9. Calculate the constant linear acceleration derived from the specified distance to stop and maximum linear speed produced at this distance by constant acceleration motion.
10. Feed the travelled distance, the leg distance and distance to stop into a formula node. The algorithm in the formula node is based on the simulation algorithm (Appendix B.7).
11. If the travelled distance is under certain threshold, give the velocity command a tiny positive value to start the motion.
12. Send the output of the velocity command to the global variable "Reference velocity".
13. Use the gradual increase of linear velocity to couple the angular clamp for the turn controller related by the global variable "DirectionControlClamp".

9 Algorithm for Figure 11.32a

This simulation algorithm changes the turn curvature (the inverse of the turning radius) along the angular distance of the turn in the same way the distance controller algorithm in Appendix B.7 changes the reference velocity along the travelled distance of the leg. The main difference is that the acceleration and deceleration transients are located at exactly 1/6 of the turn and not at a specific distance. The curvature is analytically related to the left joint steering angle.
Turn controller simulation algorithm

1. Divide the angular distance \((\text{bel} == \text{bearing error})\) of the turn (in radians) by six. The first section \((1/6)\) is devoted to acceleration transient. The last section \((1/6)\) is devoted to deceleration transient. The four sections in between are devoted to a constant turning radius motion. The algorithm strives to guarantee that \(2/3\) of the turn \((4/6)\) are done with a constant turning radius.

2. Set the robot's top linear speed and the desired turning radius.

3. Set the desired value for the distance to stop for linear motion \((x5 = 1.5 \text{ meter})\).

4. Calculate the constant linear acceleration \((a1)\) based on the desired top linear velocity \((x4 = 0.5 \text{ m/sec})\) and distance to stop \((x5 = 1.5 \text{ meter})\).

5. Calculate the constant angular velocity \((\omega2(i) = x4/rl)\) derived from the targeted radius \((r1 = 1.5 \text{ meter})\) and the targeted linear velocity of \(0.5 \text{ m/sec}\) (which yields angular velocity of \(0.5/1.5 \text{ rad/sec}\)).

6. Calculate (implemented by the function TransitionTurning.m where \(r1\) is the planned turning radius, \(\text{curvel}\) is \(1/rl\), \(\text{asl}\) is the angular distance of the transition which is \(1/6\) of the turn) the constant angular acceleration \(y1 = (\text{curvel}\omega2^2)/(2*\text{asl})\).

7. Devise a function for each of the transients that control the curvature \((\text{curve} = 1/r)\) as a function of the angular distance. (see the implementation code for function \(y = \text{TransitionTurning1}(x)\) based on two transition sections of 15 degrees each \((\text{asl} = x3/6\) in TransitionTurning1.m) out of the nominal 90 degrees for a 90 degrees turn.

8. If the curvature is equal to zero (as at the beginning of the turn and at the end of the turn) than the turning radius is 10 meter (and not infinity).

9. Convert the angular distance of the turn "\(\text{bel}\)" to radians by dividing with the factor \((180/\pi = 57.3\ldots)\).

10. Start a for loop (step 11 \(\rightarrow 17\))

11. Start to propel along the angular distance of the turn by incrementing the angular distance by \(0.001*\text{bel}\) each time.

12. If the angular distance is within the first or last transient sections - calculate the momentary curvature of the turn from the trapezoidal curvature pattern with parabolic sides (curvature versus angular distance). If the robot is travelling within the 4 sections located between the transient sections - the curvature is constant and is equal to the desired inverse turning radius.

13. Calculate the incremental linear distance using the momentary turning radius and the increment in the angular distance of the turn.

14. Calculate the accumulated travelled distance for the robot (the travelled distance for the mid-rear axle of the robot).
15. Calculate the linear velocity derived from the reference velocity paradigm (trapezoidal pattern with parabolic sides) and the accumulated travelled distance.

16. If the angular distance is within the 4 sections, then the curvature is constant \( \text{rad} = \frac{1}{\text{curve}} \).

17. Calculate the momentary angular velocity by dividing the distance increment by the momentary turning radius.

**10 Algorithm for Figure 11.33**

Turn controller algorithm

1. Reset to zero the history data of the array "Forward velocity" outside the while loop.
2. Set the digital line logic state to give computer control.
3. Start a while loop (step 4 \( \rightarrow \) 9).
4. Read the bearing error from a global variable "BearingError".
5. Calculate the absolute value of the bearing error and divide it by six (to create the angular distance of the transients).
6. Calculate the travelled angular distance of the turn by dividing the difference in the rear encoders readings by the rear axle wheelbase. Find its absolute value.
7. Feed the absolute value of the bearing error (x3), the momentary absolute value of the travelled angular distance (x1) (the amount of turn being done so far), the desired steering angle at the left joint of the Ackerman steering system (x4) (translated from the desired turning radius in the navigation map and the sign of the bearing error indicating left or right turn) and the absolute value for the angular distance of the transient section (x5) (which is also the "angular distance to stop") to a formula node that calculates the trapezoidal pattern (with parabolic sides) for the (curvature) steering angle command as the function of the angular distance in the turn.
8. Export the desired steering command via the global variable "SteeringInput".
9. If the travelled angular distance in the turn is greater than the bearing error, then stop the while loop and quit the VI.

**11 Algorithm for Figure 12.1**

Data recorder algorithm

1. Reset to zero all of the 27 global variable outside the while loop.
2. Start a while loop (step 3 \( \rightarrow \) 4)
3. Record the 27 global variables (which include the sonar CTFM 128 element estimated signature) every 250 msec to the computer's buffer. The 27 global variables are
grouped in the following order: a) Navigation Data - 6 variables. b) Motion Data - 6 variables. c) Control Data - 6 variables. d) Recognition Data - 7 variables. e) Sonar Array Data - 2 variables.

4. Terminate the while loop by boolean flag read from the global "data recorder".

5. Write all the recorded data to a data file with a designated name that bears a connection to the specific experiment.
Appendix C - Animal Navigation

Animal navigation is an exciting and very challenging research topic. A lot of work had been done to understand the wonders of our world's creation, but there are still many unsolved puzzles. It is the field of robotics which links us to the world of biology, where biologists are trying to get a closer look at phenomena and biological "devices" that modern engineering can only dream about.

1 Bird navigation using the earth's magnetic field

The use of pigeons to deliver mail between two distant landmarks prevailed for thousands of years. Our ancestors revealed the capability of pigeons to navigate accurately between two landmarks hundreds of kilometres apart. Recent research has revealed that birds are using the global magnetic field and its inclination towards the poles to track their way over oceans (with featureless environments). The subject is very interesting, especially for a person who could follow the birds migration between Europe and Africa passing above Israel in huge quantities and colourful magnificent formations. There are very interesting reports about theories and experiments in many books.

For example:

a) Animal Navigation by Talbot H. Waterman. In chapter seven, page 159 there is a paragraph named "Magnetic responses in birds".

b) Supersenses - perception in the animal world by John Downer. In the chapter "Sixth sense" page 25 there is a paragraph named "Magnetic trails" illustrating how birds are using the magnetic navigation for migration between continents.

c) Sensory ecology - how organisms acquire and respond to information". In chapter 18 "Navigation" there is a paragraph 18-1 named "Accuracy of navigation" gives interesting data about navigation of flying insects and birds. In page 443 it discusses very interesting experiments with pigeons released from aeroplanes in high altitude into an insulated valley in the high Alfs and manage to regain their direction to their home without any difficulties.

2 Bat navigation and orientation

The book "Animal Engineering" by Donald R. Griffin describes in its chapter 10 - "More about Bat "Radar" - interesting experiments regarding bat orientation and navigation using their active sonar (exactly as active radar is used for "shoot and forget" guidance of modern air-to-air missiles. The seeker head of an active air-to-air missile is both transmitting and receiving the echoes).

3 Moths and Ultrasound

Another aspect of using ultrasonic radar in animals is described in the same book (mentioned in previous chapter) in chapter 11 - Moth and Ultrasound. The recordings of the signals are amazing.
Appendix D - Simulating the aliasing effect for undersampled signal

The following simulations illustrate the aliasing effects caused by an undersampling rate. The Nyquist Frequency is equal to half of the speed of the sampling frequency $F_s$ which means that in order to observe a phenomena which contains frequency components of up to $f_1$ Hz we need to apply at least a sampling rate of $f_2$ (Hz) = $2*f_1$ (Hz). The case study analysis is composed from a single blade rotating at a constant frequency of 100 Hz. The blade is illuminated by pulse train (very short duty cycle) of light continuously generated by a stroboscope. The analysis shows the effect of the change in the sampling rate on the observation of the rotating blade by investigating three case studies with different stroboscope frequencies: 80 Hz, 120 Hz and 220 Hz. Titan's narrow beam phase array CTFM sonar has a practical spectrum range between 0 Hz to 5000 Hz ("The audio frequency" in Figure 3.1). The sonar's electronics samples the analog CTFM signal ("faudio" in Figure 3.1) at 15,000 Hz (the Nyquist frequency is 0.5*$F_s$ => 7500 Hz). Figures D.1 to D.5 simulate the investigation of a single blade rotating at 100 Hz and sampled by a stroboscope at 80 Hz. Figures D.6 to D.10 simulate the investigation of a single blade rotating at 100 Hz and sampled by a stroboscope at 120 Hz. Figures D.11 to D.15 simulate the investigation of a single blade rotating at 100 Hz and sampled by a stroboscope at 220 Hz. Figure D.16 shows 16 (each sub-figure corresponds to a 10 Hz increment in the stroboscope pulse train frequency which varies between 80 Hz to 230 Hz) estimated spectrums of the single rotating blade in the frequency domain constructed from the individual pulse trains. Figure D.17 shows the location of the peak in the frequency domain versus the stroboscope frequency. The results are very interesting!
Figure D.1 Single blade rotating at 100 Hz illuminated by a pulse train light from a stroboscope transmitting at 80 Hz.
Figure D.2 The source signal 1 is generated by a rotating blade at 100 Hz while the extracted signal 2 is created by the interaction with a pulse train light from a stroboscope transmitting at 80 Hz.
Figure D.3 The extracted signal versus the time.
Figure D.4 The pulse train generated by the interaction of the source signal with a stroboscope transmitting at 80 Hz.
Figure D.5 The normalised estimated spectrum constructed from the pulse train data.
Figure D.6 Single blade rotating at 100 Hz illuminated by a pulse train light from a stroboscope transmitting at 120 Hz.
Figure D.7 The source signal1 is generated by a rotating blade at 100 Hz while the extracted signal2 is created by the interaction with a pulse train light from a stroboscope transmitting at 120 Hz.
Figure D.8 The extracted signal2 versus the time.
Figure D.9 The pulse train generated by the interaction of the source signal with a stroboscope transmitting at 120 Hz.
Figure D.10: The normalised estimated spectrum constructed from the pulse train data.
Figure D.11 Single blade rotating at 100 Hz illuminated by a pulse train light from a stroboscope transmitting at 220 Hz.
Figure D.12 The source signal1 is generated by a rotating blade at 100 Hz while the extracted signal2 is created by the interaction with a pulse train light from a stroboscope transmitting at 220 Hz.
Figure D.13 The extracted signal2 versus the time.
Figure D.14 The pulse train generated by the interaction of the source signal with a stroboscope transmitting at 220 Hz.
Figure D.15 The normalised estimated spectrum constructed from the pulse train data.
Figure D.16 16 normalised estimated spectrums generated by a 10 Hz increment in the stroboscope pulse train frequency.
Figure D.17 The position of peak of the estimated spectrum in the frequency domain versus the stroboscope pulse train frequency (80 → 230 Hz).