Undergraduate Research Report
A Comparative Analysis of Modelling Mechatronic (Hybrid) Systems

Chinemelu Ezeh
Department of Electronics and Electrical Engineering
Imperial College London

December 30, 2013

"If you really want to understand how something works, build it yourself..."

Contents

1 Introduction

Mechatronics can be defined as the synergistic combination of mechanical, electronic and software engineering [2]. It is recent and consequently an underdeveloped field that has a major challenge of establishing a standard in design principles that could be appreciated by engineers from various domains. The purpose of these design principles is in modelling hybrid systems so as to maximise available resources in the presence of external factors such as the available manufacturing tools, government regulation, market demands and conditions. In this paper we will focus on analysis the modelling of hybrid systems in order to extract these design principles, using an autonomous vehicle as case study.
Modelling here is defined as any work that can be used in understanding and making decisions about the mechatronic system. By this definition, modelling begins right from analysing the client's requirements to analyses of the final product so it takes into account the entire manufacture process. Thus it is important to highlight and discuss the entire mechatronics design process. Traditionally, this is done sequentially using various high level process models such as Meisler's Conceptual to Rodenaker's Structural shown below [2].

Meisler's Conceptual Model	Rodenaker's Structural Model
1. Formulation of the design	1. Clarify task
2. Generation of Alternative Design Solutions	2. Establish function Structure.
3. Analysis and evaluation of alternative	3. Choose physical processes
4. Selection of preferred alternative	4. Determine embodiment
	5. Check logical, physical and constructional relationships
	6. Eliminate disturbing factors and errors
	7. Finalize overall design
	8. Review chosen design

Traditionally, design is done sequentially and this means factors like testability and manufacturability are not taken into account at the early stages. However as systems become more complex, this design method leads to unforeseen problems in the later stages due to incompatibility since constraints in electronics and software were not taken into account from the beginning of the design. The mechatronic community has agreed that any design model must take into account all elements of the design life cycle from the incipient stages beginning from concept to disposal including quality, cost, schedule and requirements. This is called concurrent engineering and is shown in the figure 1.

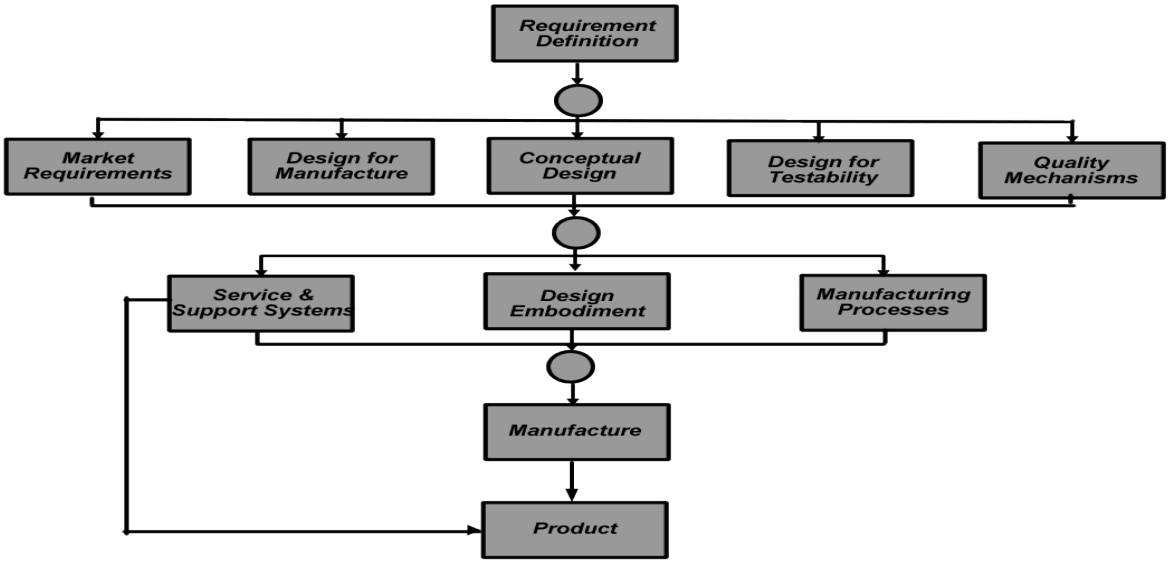

Figure 1: Concurrent Design Process

There are a number of tools available to support the concurrent engineering process. These tools can

be though of as perspectives or view for reasoning about the design of mechatronics. Engineers from different domains focus on different parts of the hybrid system so by using a family of tools/models, hybrid systems can be designed concurrently as any single view is not sufficient .These models include:

1. Quality Function Deployment (QFD): QFD is aims at considering customer inputs are part of the specification process. It involves four sequential phases. First the requirements are converted into quality characteristics of the system after which a planning is done to define the characteristics of the critical parts. This planning can be done as a matrix chart which has on its left side the requirements and along its top, the product features and functions (outputs). The relationship between inputs and outputs are classified as strong, medium, weak, none.
2. Robust Design:
3. Failure Mode Effect and Criticality Analysis:
4. Design for Manufacture and Assembly:
5. Design for Testability, reliability and Service:

In practical however, a model must be able to adapt, or change, to suit a particular task at hand. A proposed flexible framework for a practical design process is based around four interdependent absolutes:

1. People
2. Processes
3. Tools
4. Information

The design of a multi-disciplinary complex system relies heavily on the design team. The diversity of technology/processes available makes this design possible. Nevertheless, a good team must have some common tools with which everyone is comfortable with. The team also needs to share information as it is the maturity of information that allows for successive progression in the design process. Implementation of the actual design process will be unfolded as we go along.

2 Requirement Interpretation

An important part of modelling a hybrid system is to first understand and interpret the user requirements into design specifications. Here, we are concerned with what the system does and not how it does it. There could be hidden requirements that can easily be omitted in the design process. Moreover, requirements may not consider the whole life cycle of the design. According to Taguchi loss function, any product that does not meet its target specification will either impact a loss to the company or a loss to the society. We will look at this concept and how it affects the modelling process in due course. (**For scalability, what parts of the model needs to be abstracted and what parts need to be in focus?)** In general, requirements can be divided into functional and non-functional. The functional requirements are those that specify the functions of the system, i.e. changing these will change what the system is supposed to do. Non-functional requirements detail constraints, targets or control mechanisms for the new system. They describe how, how well or to what standard a function should be provided. For example, levels of required service such as: response times; security and access requirements; technical constraints; required interfacing with users' and other systems; and project constraints such as implementation on the organisation's hardware or software platform. Service level requirements are measures of the quality of service required, and is crucial to capacity planning and physical design. Identify realistic, measurable target values for each service level. These include service hours, service availability, responsiveness, throughput and reliability. Security includes defining priority and frequency of backup of data, recovery, fall-back and contingency planning and access restrictions. Access restrictions should deal with what data needs protected; what data should be restricted to a particular user role; and level of restriction required, e.g. physical, password, view only. Non-functional requirements may cover the system as a whole or relate to specific functional requirements. While it may seem that there is a clear distinction between functional and non-functional requirements, an issue is that when treating a requirement such as performance, does one model it as a functional or non-functional requirement?

2.1 Categorising Requirements

The steps involved in interpreting requirements are:

1. Textual analysis: The purpose of this is to expand on the requirements so as to produce a specification that is suitable for further analysis. A simple approach is to first list all the requirements of the system and then separate these into functional and non-functional requirements. Thus for our vehicle, we have,

   Functional Requirements:

   - Ability to detect nearby objects.
   - Be able to track its motion.
   - User interface to issue commands.
   - Be able to keep track its position on a map.
   - Ability to generate this map of its environment through exploration.
   - Be able to navigate automatically from one point to another.
   - In the event of an error, the system must be able to fail gracefully.

   Non-functional Requirements:

   - Vehicle should be toy-size to serve as a prototype.
   - Hybrid system must be scalable.
   - It must function properly most of the time.
   - Design should minimise energy consumption.

   With this list, we may now ask what is missing in the requirements. What about reliability and safety? Since this is an iterative process, we may need to answer these questions as we go along. For now however, the structural relationship between the functional requirements need to be clarified using the viewpoint analysis together with functional analysis and modelling.

2. Viewpoint Analysis [2]: This is based on the notion that given a complex system, only a partial understanding will be gained from any single point of view. Thus a software engineer will be concerned about the information processing of the vehicle while a mechanical engineer will be more concerned about physical structure of the system. There are two type of viewpoint analysis: functional and non-functional. Functional Viewpoint supports partitioning the system logically into modules that transform information. There are two types:

   (a) External viewpoint provides an external view of the system i.e. how the world sees it.
   (b) Internal viewpoint presents an internal view of the system.

   The non-functional viewpoints represent a group of requirements that modify or constrain the function requirements of the system. The first step in structuring the functional viewpoint is to identify and group the bounding and defining viewpoints. The next step is to form a hierarchy chart from this information. Thus for our vehicle we have,

   Internal viewpoints:

   (a) Localisation
   (b) Path Planning
   (c) Navigation
   (d) Exploration/SLAM
   (e) Map building

   External Viewpoints:

   (a) User interface

   These can be structured as shown below:

(a) Functional Analysis and Modelling: It is important to develop a functional model of requirements as this will form the basis for conceptual design that we shall perform later. A good modelling method must be able to capture both hardware and information features. It must be able to include human and non-technical interactions and be able to present information in a clear and simple way. There exists modelling techniques such as CORE, SADT, SSADM, VDM and IDEF. These techniques however require specialisation and thus system designers and other non-technical contributors are not able to contribute. An alternative model, one that is simple enough should be favoured instead. A candidate is the DeMarco model which uses diagrammatic methods that encourages the separation of function from non-functional requirements. There are three main parts of the DeMarco model: a) The Data-flow diagram that portrays the system in terms of its activities and the information flow between these. The main elements are the data-flow which is a vector indicting the information flow, the process which transforms incoming data-flow to outgoing data-flow, the terminators which are sources or sinks of data external to the system and finally data files which are temporary storage for data. b) Data dictionary holds the description of every data-flow and data file. b) The process specification holds the description of the processes.

Thus we can generate a model such as,

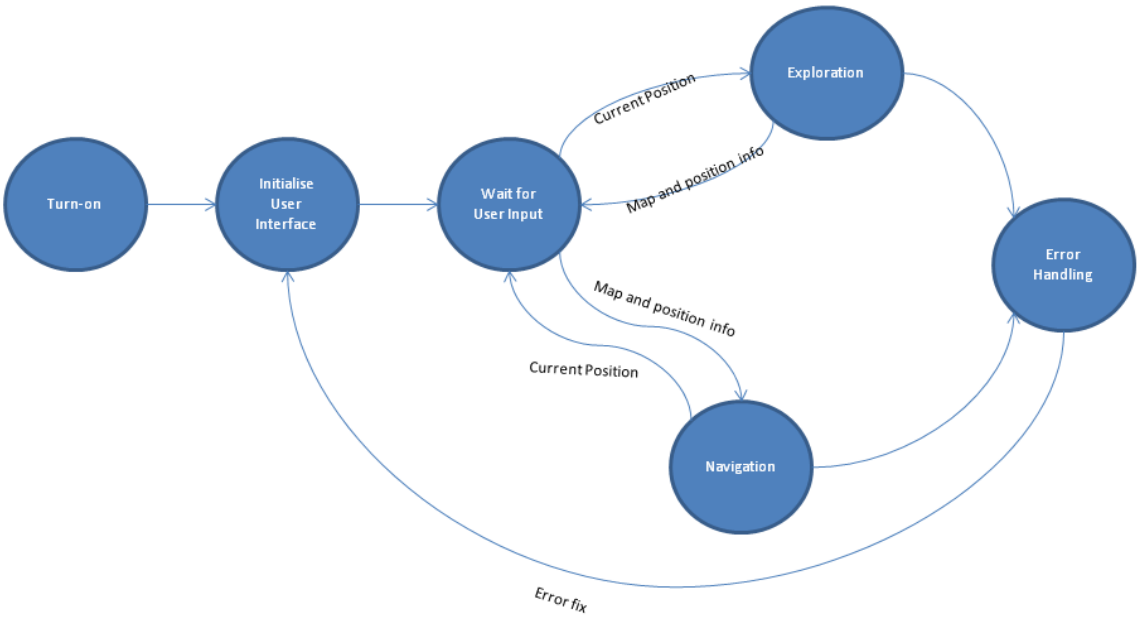

Figure 2: Functional Model of Requirements

3 General Software Development Process

Software is the most complex part of the hybrid system. The reason is that most of the self-adaptivity is in the flow and processing of information in order to control the system's response to its environment. By software, we are referring to areas of the system from the high level of abstractions such as path planning, overall system manager to the lower levels such as control systems down to periphery drivers. As with the entire system, there are a few approaches in modelling the software design process. We however need to specify some terminologies that will appear in this section. These are:

Process	Description	Datafile	Description
Turn on	This represents the start of the system	Current Position	This represents the coordinate and orientation of the vehicle in the map
UI Initialisation	Starts the user interface which displays information on the vehicle such as battery level, map(if available), robot position, explore and navigate commands.	Map Information	This represents an array or data structure containing points that define the map boundaries
User Input Wait	Processes user's options to either browse UI or Control Vehicle either manually or automatically.	Error Fix	This represents the solution or response to error handling
Exploration	Manipulate the vehicle either manually or automatically so as to generate a map of the environment using SLAM.		
Navigation	Robot moves autonomously from a start to an end destination		
Error handling	Manage any errors that may arise. This could be an error in data connection or an error in algorithm or anything. Explicit error handling is done when an unknown error is caught		

Figure 3: Process Description and Data dictionary

1. Component: This is a constituent element of the system which may have one or more functionalities depending on the context. A component can be software, electronic or mechanical part. For example, a motion encoder is a component whose primary job is to provide data about the motion. Its however may either be to output a trajectory of system or to provide feedback for motor control. A software component can be a path planner programme which based on a vehicles perceived environment will either choose to exploration or navigation depending on whether the robot has sufficient knowledge of its environment.

2. Interface: An interface is a point of interaction between two components. Interfaces have ports which ensure that only intended interactions take place. An interface will also have a communication protocol which specifies how the two components will understand themselves. For example, a laser range scanner only provides raw data. This data needs processing so it is to be sent to a processor. An interface is needed to specify details such as how often data will be sent, what data structure will be used, the wire connections (port) between the two components, etc.

3.1 Requirement Analysis

Here, the aim is to define the high-level design of the system through the analysis of requirements of the software. There are several approaches in specifying this high level architecture. One can use a structural hierarchy of system components including hardware. A structural view specifies the connections/interfaces between the system components. Here, the non-functional requirements are not taken into account explicitly. Another suggestion is in using a structural, behavioural and management 3-view model [6]. The structural view separates required interfaces and provided interfaces as this allows for easier analysis of binding required and provided interfaces together also know as configuration management. The behavioural view consists of different models of the system's component functionality and models for non-functional concerns. The management view comprises of the management of functional component which provide the means for adaptation and the management components themselves. This 3-view model separates the system design into perspectives that have very few overlap, thus these are effectively orthogonal perspectives that allow for modularity in design concepts. Nevertheless, there is still some overlap amongst the views. The management components have functional interfaces which will appear in the structural view as interfaces and in the behavioural view and functional components.

The advantage of this approach over the others is that it explicitly attempts to model the non-functional requirements which can be equally important in the design process. For example, it is a good idea to design an autonomous vehicle taking in account how best to minimise its energy and CPU consumption as resources are limited. Modelling these requirements explicitly lets us better manage complexity in the design process as we now define formal syntax and models for them. The question that remains however is, "where to draw a line in this overlap for an optimal design?" We will explore this along the way.

Structural Model Here we are concerned with separating the system's components along its interfaces focusing on the high-level abstraction of the system's software. In the high-level, we have the user-interface from which a user can monitor and control the hybrid system. From the UI, a user can either choose to create a new map of the environment by choosing the exploration option or he can choose to navigate on a known map from one point to another. Another component of the UI is to monitor the power level(not shown in diagram).
The main components of our system are:

- UI: This provides display to view map with vehicle location included as well as power level and a capability to control the robot to navigate from one point to another or to build a map through exploration.
- Path Planning: This takes an input the current location in order to generate a path to a desired location. The robot is then controlled autonomously to goal location.
- Explorer Module: Builds a map of environment using the gmapping module and control system.
- Gmapping: This module provides the core algorithms based on simultaneous localisation and mapping (SLAM) that would be used by the explorer in building a map of the environment.
- Control and Feedback System: This provides a high level control to the sensors and actuators in order to manage their operation. Its tasks include enabling and disabling the actuators and sensor drivers, and specifying changes in parameters such as vehicle speed.
- Motor Controller: is responsible for processing the motor encode signal and the commands from the control system in order to provide continuous signal to the actuators or to smooth transition between changes in the speed for instance.
- Scanner Controller: is responsible for controlling the actuators that move the laser scanning sensor based on information received by the control system and the signals coming in from the laser range scanner that it also processes.

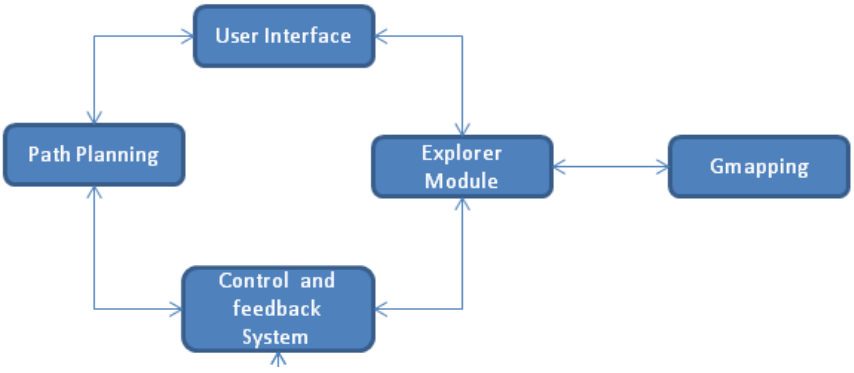

Figure 4: High level Structure

Functional Model The functional or behavioural model is used to describe the behaviour of the system as it evolves over time. In the high level, we will focus on the core behaviour of the system such as the behavioural interaction between components for example how the UI will interact with the path planning module. We can show the behaviour of our system starting at the user interface in the figure below.The behaviour of the explorer and navigator can also be expanded to give details of their functions. Here

we expand on the compute path to goal behaviour. Here we show the explorer module and an auxiliary module called gmapping which produces the actual map and localisation information of the system.

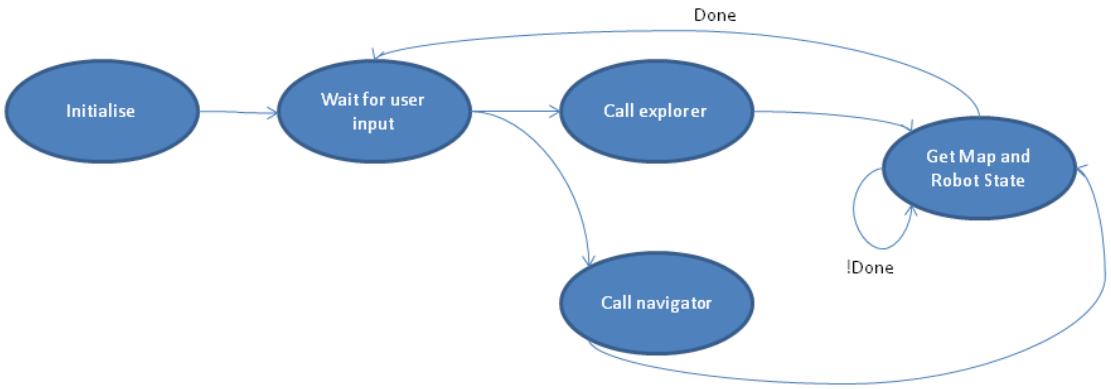

Figure 5: UI Behavioural Model

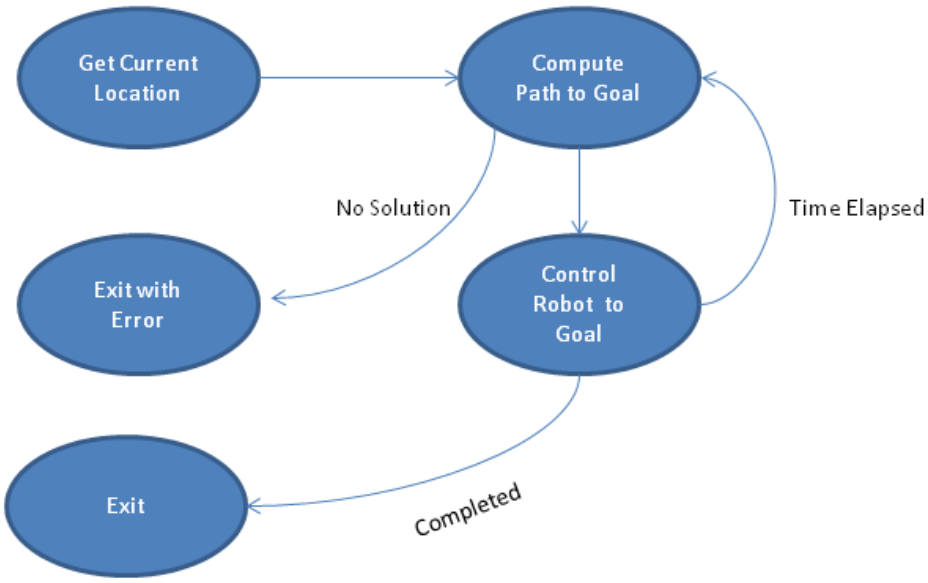

Figure 6: Navigator behavioural model

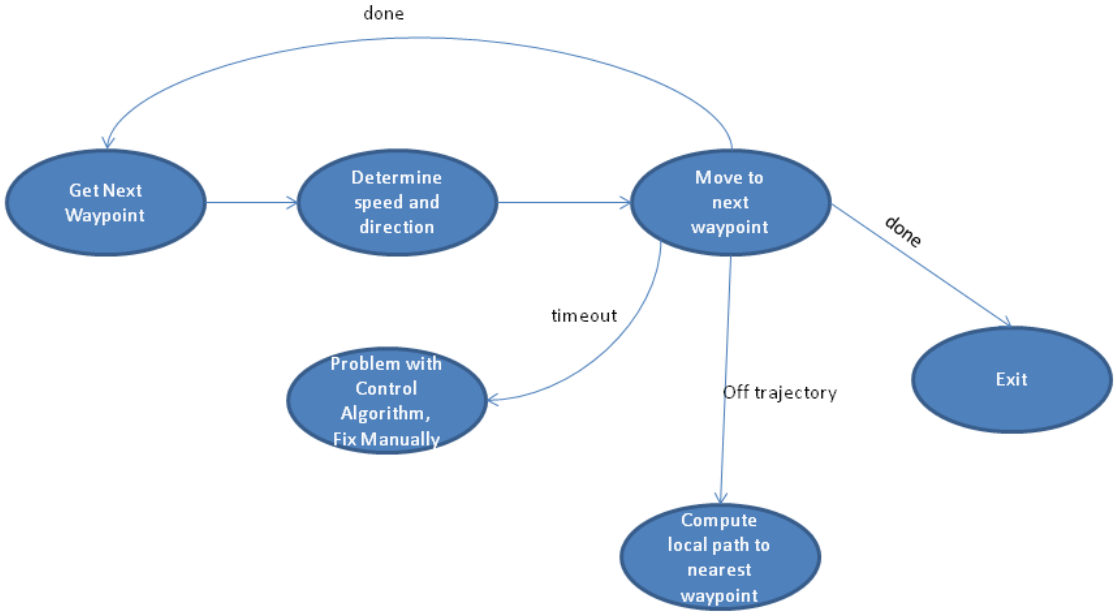

Figure 7: Path Computation Behaviour

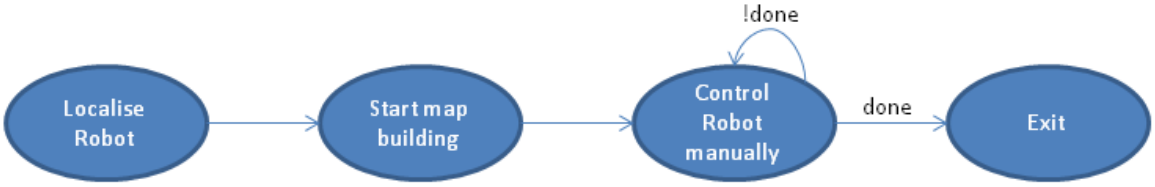

Figure 8: Explorer Behaviour

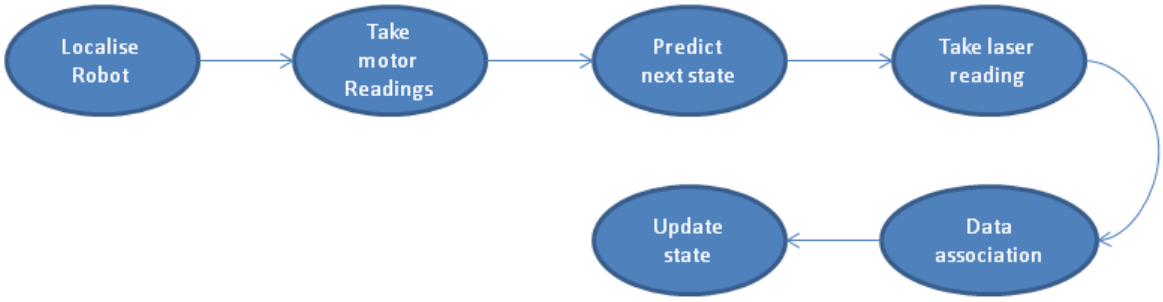

Figure 9: Map builder (Gmapping) Behaviour

4 Approaches to Mechatronic Hardware Design

In this section we will look at what hardware design is all about. By hardware, we are referring to the mechanical, electronics and electrical parts. There is a trade-off of functions between these, i.e. how

does one start the process? What are the principles and best practices? What are the limitations with existing technology? What are the tools?)
There are several approaches to designing the hardware. This hardware includes the mechanical, electrical and electronic parts of the system. The mechanical domain primarily deals with rigid body systems while the electrical domain deals with the relationship between voltage and current. Electronics involves applying electrical principles to process information and develop devices. In this regards, the demarcation between mechanical and electronics and electrical is clear. There are a few tools such as AutoCAD for 2-D and 3-D mechanical modelling. Briefly, the modelling process of mechanical systems involve specifying the product size and material, analysing the effects of forces on the system through simulations or otherwise. For electrical systems, one can use tools such as Lab View and Matlab for rapid prototyping of electrical systems in a high level of abstraction.
In general, there are a few features a good design should have:

- Be generic and not specific to a particular field
- Facilitate the search for optimum solutions
- Not constrain inventiveness and experience
- Be compatible with other disciplines
- Not rely on chance
- Facilitate the application of existing solutions to related tasks
- Be compatible with electronic data processing
- Be easily taught and learned
- Allow for objective evaluation
- Encourage creativity and intuition
- Reflect Modern management thinking

A general procedure for integrating the systems is

- Pure mechanical system.
- Addition of sensors, actuators, control functions
- Integration of components (hardware)
- Integration of information processing(software)
- Creation of synergistic effects to give the final system.

For our vehicle, the pure mechanical system was given to us in a ready made state. Our modelling approach would be to establish the electrical and electronics connections based on our understanding of the requirement. This can be represented using graphs based on structural modelling. This approach is in line with concurrent design because we will be considering the implications of our model on software development. The reason for this approach is that electrical connections are structural in nature so for a simple model we will abstract our design focusing on the connections. The exact placement of this connection in the physical system can be done in a simulation using tools similar to AutoCAD for very complex systems but this is not necessary for our system as our requirements are not strict about wirings. We need a user interface to view the properties of the system monitoring, perform some error handling and to interrupt in the autonomous process in areas such as manually deciding if the map generated is complete. We can choose to have this user interface on a laptop for the flexibility of interface development and processing power for software development, it will provide. This laptop can be our main processor and would be in-charge of the high level functionality and processes that need considerable processing time. On the vehicle, we still need a processor on the system to interface with the sensors. Thus for a communication between the two sensors, we will use a wireless connection. This is shown below:
Note: In our case study, our design is not sensitive to the placement of sensors. The motor encoders need to be placed near the the motors and the optical sensors need to be placed in a region that allows for

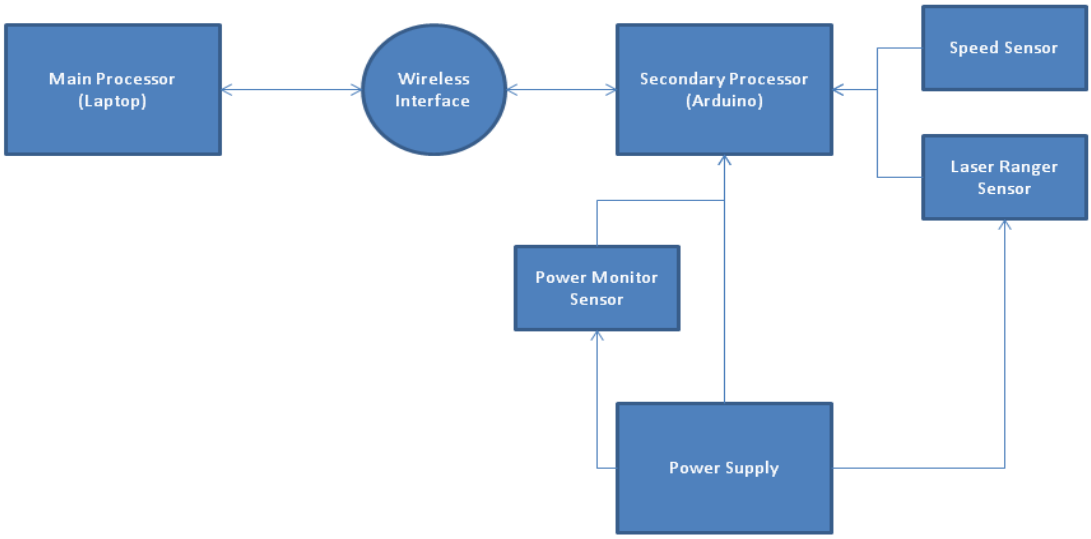

Figure 10: High Level Electronics Connection

360^0 range of movement or at least 180^0. This would mean the sensor should be placed anywhere of the top of the vehicle or on in front and the other in the rear. Corrections in the exact position of the sensors can be corrected using a translation matrix relative to the centre of our vehicle. The optical sensors provide information of a 2d world similar to a cross section of the environment along the horizontal axis to the vehicle. This means information of the three dimensional properties of the environment is lost but we can assume a simple environment in wwhich this design is sufficient.

5 Overall Architecture

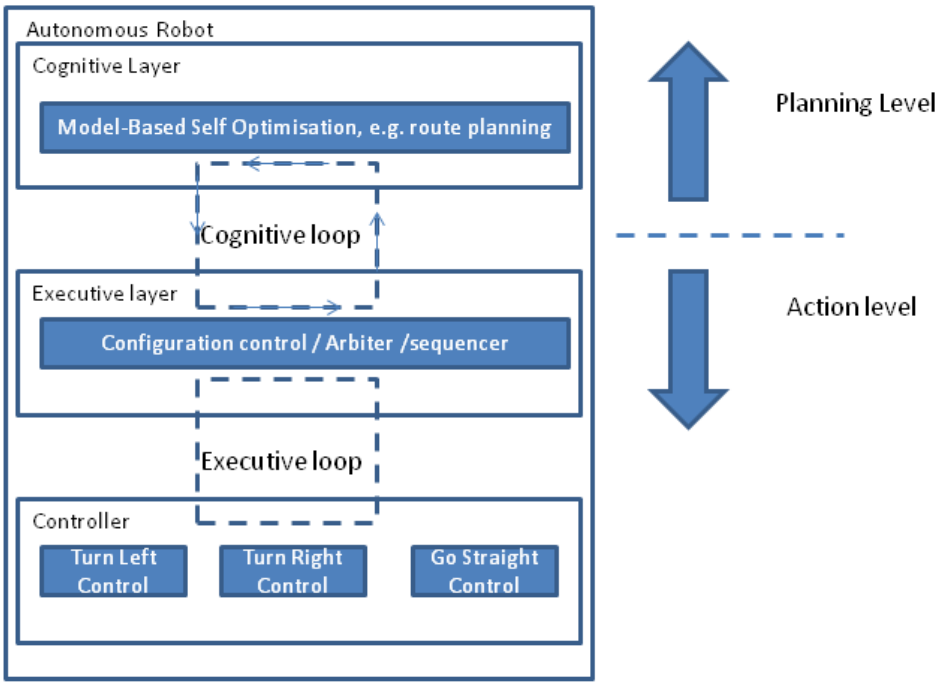

Figure 11: The overall architecture of vehicle as a mechatronic system

Using our understanding of the system based on the requirements, we can divide the overall system into 3 level of hierarchy[4]. At the top layer is the cognitive layer. Here the system gathers information on its self and its environment and uses it to alter its behaviour in performing long term tasks which are possibly time consuming such as path planning and thus, this level introduces intelligent behaviour. The lower level is the executive layer which is predominantly event-driven. It executes control and monitoring routings but does not interact directly with actuators and sensors of the vehicle. This layer is essentially a discrete system that comprises of control automaton responsible for managing the functions on the lowest layer called the controller. This is the continuous part of the system and it implements the control signals to the actuators and provides the feedback loop based on sensor data. Signal processing is also done on this level to provide smooth transition between control strategies such as changing directions or speed of the vehicle. Due to the different natures of the layers for example, the controller layer is a continuous system unlike the executive layer which is a state machine, there is a need to investigate the ideal modelling approaches and tools for the various layers. For instance, using a state-machine notation for the controller layer's function may not be ideal since the mathematical nature of the continuous system is incompatible with this notation. The layers interact in a feedback manner denoted by a loop abstraction as depicted in figure 11. This interaction involves issuing commands to and fro the layers to make more intelligent decisions from the communications between the layers. We would delve into details on the architecture in section 7.5

6 Design Tools, Their Pros and Cons

Before proceeding further, it is beneficial to get a grasp of the available tools at our disposal for modelling hybrid systems in general. Currently there isn't a complete all-in-one solution but several software can be used for different parts. In this section, we would look at these software and discuss their applications and limitations.

1. UML: is very useful for the software systems. It was in fact designed for this purpose especially in representing requirements using diagrams and connected graphs. With UML one can develop a high level description for behavioural, structural or parametric models. On the other hand, UML's

specialisation in software means that its abstraction is not compatible to other domains such as mechanical and electrical engineering. This is very visible in attempting to model continuous systems. UML does not provide specific formal syntax for components outside software domain. In fact, UML is useful only for discrete systems. Moreover, there are no timing diagrams and time based graphs and simulations to test and verify models. This means that UML if applied to other domains can only be used for the high level modelling of system structure and behaviour(State transition diagrams, etc.) and can not provide detailed models specific to the domain.

2. SysML: Unlike UML, SysML attempts to include other domains in its syntax. SysML reuses a subset of UML 2 to provide additional extensions needed to address requirements in the UML for Systems Engineering. **How are these requirements address?** In any case, SysML still suffers from the same fundamental limitation of UML in that it is too abstract for components of other domains and there is not formalisation of continuous systems and time notation.

3. OpenModellica: This is an impressive modelling tool. It attempts to incorporate the mathematical descriptions of notations, concepts and components from all domains of engineering. It is even able to combine smoothly component from different domains in order to develop a system that can in fact be run on a simulation and tested for verification of operating conditions. Modelica has a graphical interface that shows the connections of components while it still retains the notations of components as they have been used in domain of origin. Modelica seems to be the way forward in modelling hybrid systems as it is still underdevelopment. Areas susceptible to improvement include the fact that this tool is not for rapid prototyping as it follows a strict syntax similar to C. So to develop complex systems, one has to code this from near scratch down to the minute details of the model. There are libraries that have models of complete components such as a car engine, etc. There however needs to be more libraries available so that the modelling process can be reduced to combining already existing blocks/components in order to allow for rapid prototyping. Thus users are freed from worrying about the detailed principles of deriving the models in order to focus on fitting the overall system together in an optimal way. Also, providing a database containing model knowledge will also help in rapid prototyping. This knowledge base can be compared to Matlab toolboxes that show descriptions of components and functions as well as how these can be used with other components. Unlike UML, it is not easy to deduce the behavioural flow of information as there are no state transition charts. This is useful when one needs to get an abstract overview of how a component works. Researchers have mentioned a problem of integrating multiple domains to establish theoretical models because theories from different domains may be based on different axioms. I am yet to see how this affects Modelica.

4. LabView: provides a comprehensive environment specialised for electrical and control systems. It can be used to model and simulate continuous time systems unlike UML. However, it can not be used for modelling the complete hybrid system only a part of it.

5. Simulink: Similar to LabView, simulink provides rapid prototyping for signals, control and dynamic systems. It is not used for mechanical or software modelling.

7 Context Investigation

In this section based on [7], we aim to investigate the principles behind modelling the hybrid taking into account that the various layers are different in function and characteristics. Nevertheless, these layers must also interact. Thus we will also look at how to model this interaction as well. It appears that at different layers, different modelling approaches are need. For example at the controller level, mathematical models of control systems would be appropriate for defining the actuator's behaviour while at the executive layer there may exist a state machine which is probabilistic in nature generally speaking. At the core of our hybrid system should be the ability to navigate autonomously. Thus this appears to be a good sub-module to start investigation in. For navigation to be possible, we require that the control system, perception of environment, localisation and path planning models to be established. The control system model tells us the possible types of movements and trajectories of our vehicle. The robot must be able to perceive its environment if it wants to navigate through it. The perception model gives us information about what information we can deduce from the environment and its associated quality describing range and errors in measurement. Using its perception, the vehicle can form a belief about its current location in the map. The algorithm needed is detailed in the localisation model. Finally, the

path-planning model combines all the previous models to plot a route on the map that the object can follow based upon its movement capabilities and its localisation. The robot can then follow the path by actuating the motors in such a way that it follows the predefined path, perceiving its environment to provide periodic feedback for its own localisation. Below is the basic overview of our navigation system.

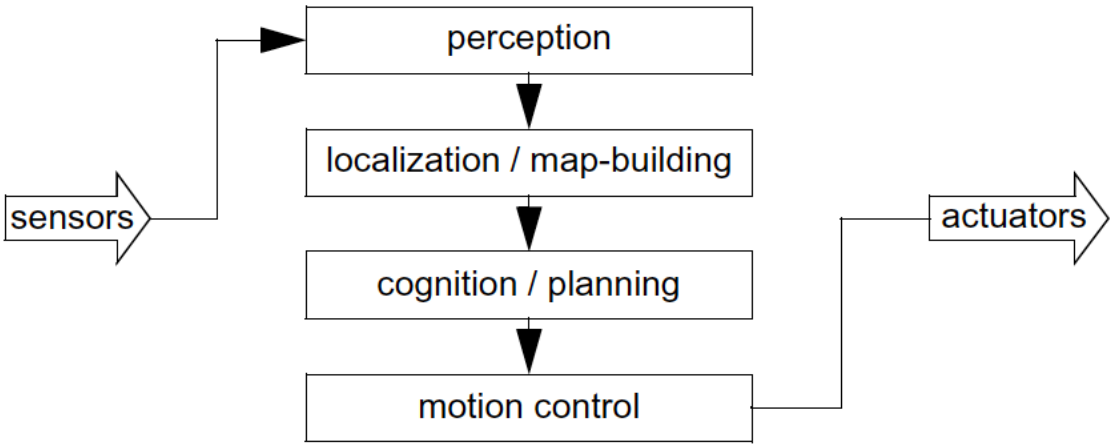

Figure 12: map based navigation

First however, we need to establish a mathematical model of the vehicle itself as a mechanical system. Using the kinematic properties of the system, we can then establish a control system model. Once we have established the core model of the system, we can proceed iteratively to model the entire system by adding other aspects.

7.1 Kinematic Model

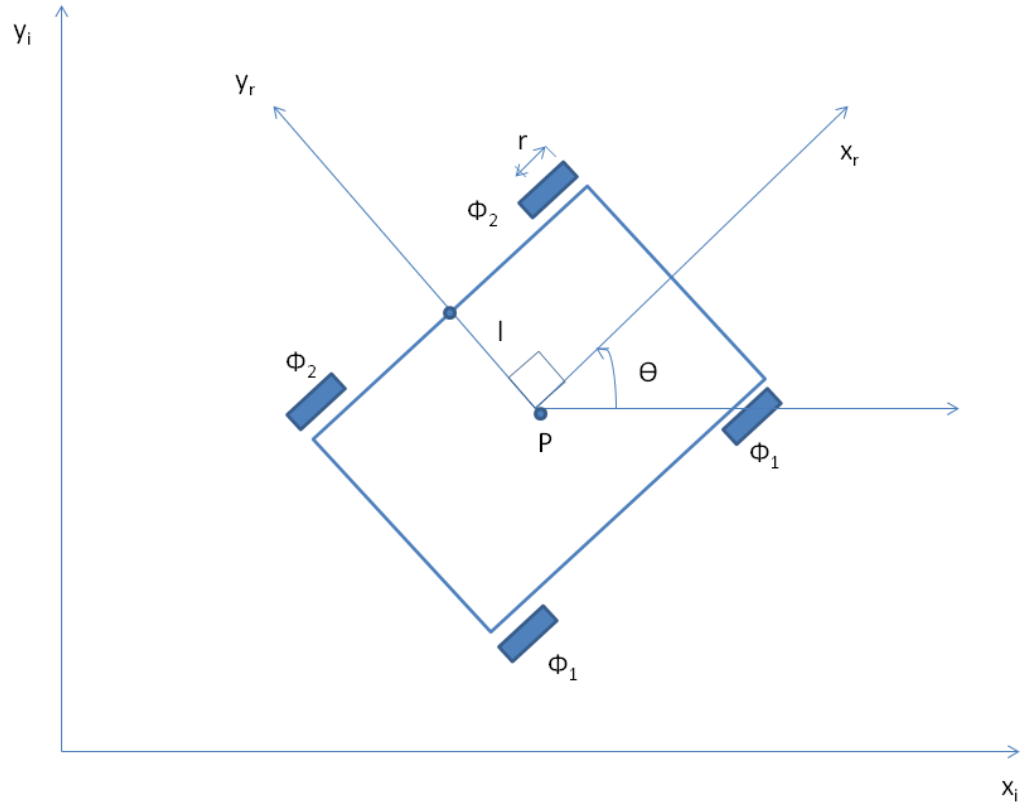

Figure 13: 2D graph of robot kinematics

Modelling of an autonomous vehicle is done from bottom to top. Thus we must first understand how the mechanical system will move based on the movements of its differential drive wheels. Here the speed of the right wheels are always the same and can be set independently of the left wheels. For our model, we have assumed that our environment will be completely flat and that the wheel will not slip or slid. Thus by considering how each rolling wheel affects the robot's motion, we can derive a forward kinematics model of our system to be give as

$$\dot{\Upsilon}_I = R(\theta)^{-1} \begin{bmatrix} \frac{r\dot{\phi}_1}{2} + \frac{r\dot{\phi}_2}{2} \\ 0 \\ \frac{r\dot{\phi}_1}{2l} + \frac{r\dot{\phi}_2}{2l} \end{bmatrix} \tag{1}$$

Where $\dot{\Upsilon}_I$ represents the instantaneous change of the state matrix Υ_I of our system and our state contains the coordinates of P and the angle θ between the robot basis and the world frame basis. Mathematically,

$$\Upsilon_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{2}$$

$R(\theta)$ represents the translation matrix of a point in the world's frame of reference to the robot's frame of reference. ϕ_1 and ϕ_2 are the rotational speed of the right wheels and left wheels respectively. The radius of each wheel is given as r and the distance from the centre at P to the line crossing either the two left or right wheel is given as l.
This approach can tell us about the robot's motion given its wheel speeds in most straight forward cases. However, we want to know the space of possible motion and for this we have to take a different approach to consider the rolling and sliding constraints imposed on fixed wheels. We will consider the contribution of each of $\dot{x}$, $\dot{y}$ and $\dot{\theta}$ to the velocity of a wheel. This gives,

$$\begin{bmatrix} sin(90°) & -cos(90°) & lcos(90-\alpha) \end{bmatrix} R(\theta)\dot{\Upsilon}_I = r\dot{\phi} \tag{3}$$

Here l represents the length of the line from the centre of each wheel to the origin of the robot chassis at P. α is the angle between this line and the x axis of the robot. Every other notation is the same as described previously. The sliding constraint enforces that motion orthogonal to a wheel's motion must be zero. This is represented as:

$$\begin{bmatrix} cos(90^\circ) & -sin(90^\circ) & lsin(90-\alpha) \end{bmatrix} R(\theta)\dot{\Upsilon}_I = 0 \tag{4}$$

We can combine all the rolling constraints of all wheels.

$$J_1 R(\theta)\dot{\Upsilon} - J_2\phi = 0 \tag{5}$$

J_1 is a 4×3 matrix denoting the rolling constraints of the four wheels. J_2 is a diagional 4×4 matrix containing the radii of the wheels and ϕ is a 4×1 matrix containing the rotational speeds of the wheels. Similarly for the no sliding constraint we have,

$$C_1 R(\theta)\dot{\Upsilon} = 0 \tag{6}$$

C_1 is a 4×3 matrix denoting the sliding constraints of the four wheels. Thus, we can combine the constraints to give an equation that defines the entire kinematics of the robot chassis. We have,

$$\begin{bmatrix} J_1 \\ C_1 \end{bmatrix} R(\theta) = \begin{bmatrix} J_2 \\ 0 \end{bmatrix} \tag{7}$$

(**It would be interesting to find software tools that can be used in deriving these equation.** Perhaps the tool maybe a 3D modelling tool where constraints can be rapidly added and and its effects simulated for verification.)
By giving different values for the motor speed, we can run simulations on openModellica to see the path of motion accrue to our speed values. This model will later be combined with control laws to derive a controllable system we will use in developing a navigation system for our robot. From our system, we can intuitively see that our vehicle has two degrees of freedom. That is it is able to change its pose and position along its x-axis independently. It can not change its position along its y-axis independents as this would violate the sliding constraint.

7.1.1 Mobile Robot Environment

The mobile robot will be moving in an environment with the assumption that this environment is flat. We need to situate our analysis of the robot's manoeuvrability on the actual environment since we want to know how the robot can use its degrees of freedom to position itself within an environment. For example, even though our vehicle has 2 differential degrees of freedom, it can position position itself at any x-y plane and at any angle. Thus is has three degrees of freedom in its environment. With this in mind, there is a need to investigate the types of movements the robot can perform and the possible range of trajectories available to it. This information will be important in defining control and path planning algorithms. In defining the workspace/environment of the robot, it is useful to first examine the velocity space of the robot given the kinematic constraints. This is the number of independently achievable velocities and can be represented by two axes, the forward speed, v and the change in orientation $\dot{\theta}$. The question here is with its velocity space, what are the available state the robot can be in. Clearly, it can be in any state since any point can be reached by changing the forward and angular velocities and this is a desirable characteristic. However, different motions will require different times and therefore energy for completion and thus a study of the trajectories are important as well.

7.1.2 Path and Trajectory Consideration

We care not only about the robot's ability to reach required final configurations but also about how it gets there. Consider the issue of a robot's ability to follow paths: in the best case, a robot should be able to trace any path through its workspace of poses. Clearly, any omnidirectional robot can do this because it is holonomic in a three dimensional workspace. Unfortunately, omnidirectional robots must use unconstrained wheels not like the fixed differential wheel we are using. Additionally, non-holonomic constraints might drastically improve stability of movements. Consider an omnidirectional vehicle driving at high speed on a curve with constant diameter. During such a movement the vehicle will be exposed to a non-negligible centripetal force. This lateral force pushing the vehicle out of the curve has to be counteracted by the motor torque of the omnidirectional wheels. In case of motor or control failure,

the vehicle will be thrown out of the curve. However, for our car-like robot with kinematic constraints, the lateral forces are passively counteracted through the sliding constraints, mitigating the demands on motor torque. But our vehicle also has a high manoeuvrability in the workspace with 3 degrees of freedom(DOF). So, how does this compare to an omnidirectional robot? The ability to manipulate its forward velocity and orientation independently means that our vehicle can follow any path in its workspace. More generally, any robot with 3 DOF can follow any path in its workspace from its initial pose to its final pose. An omnidirectional robot can also follow any path in its workspace but there is still a difference between a degree of freedom granted by steering versus by direct control of wheel velocity. This difference is clear in the context of trajectories rather than paths. A trajectory is like a path, except that it occupies an additional dimension: time. Therefore, for an omnidirectional robot on the ground plane a path generally denotes a trace through a 3D space of pose; for the same robot a trajectory denotes a trace through the 4D space of pose plus time. Our robot can not however follow any trajectory.

7.1.3 Beyond Basic Kinematics

Above considerations are only an introduction to the kinematics of the robot. Besides the theoretical kinematics constraint, there are dynamic constraints in practice as high speed vehicles invariably break the kinematic constraints. This appears as skidding and sliding of wheel in real life and should be explicitly modelled. Also, the robot must have proper actuation over its degrees of freedom in order to move as desired in its workspace. This is the problem of motorization. In addition to this, there is the problem of controllability which poses the question of under what conditions can a robot go from its initial position to its final position in bounded time. It turns out we need knowledge of both the kinematic and the control systems that can be used.

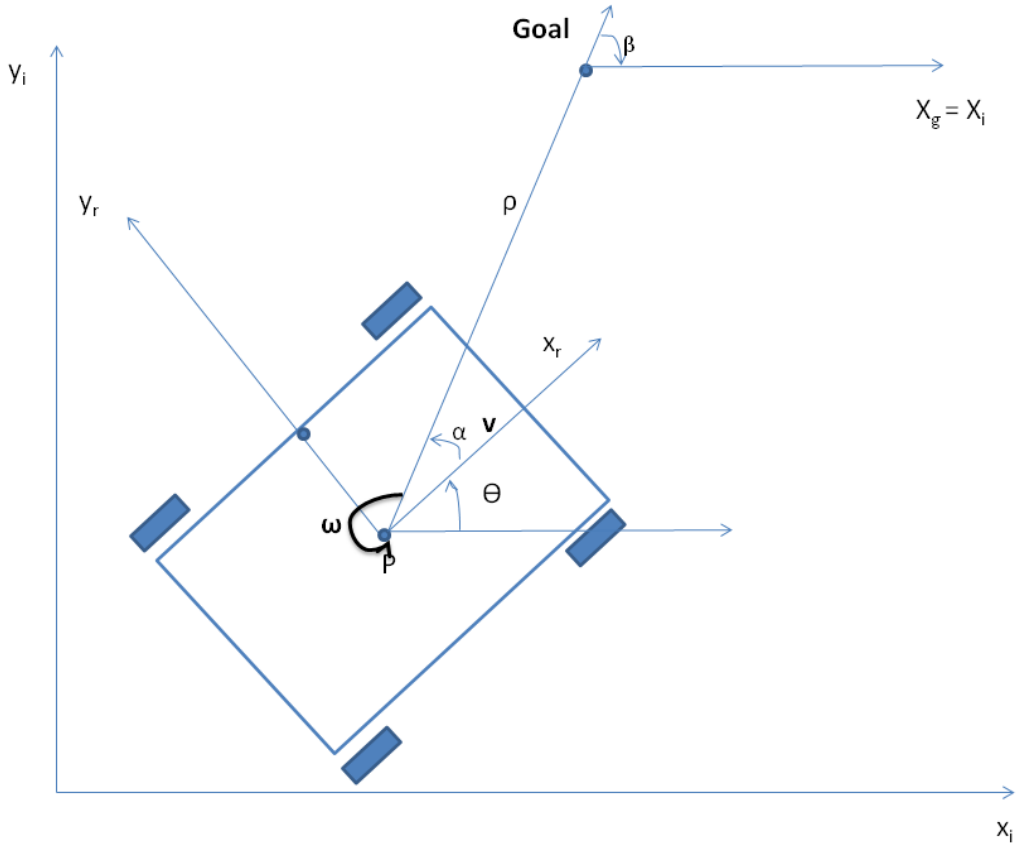

Figure 14: Robot kinematics and its frame of interest

7.2 Control System

Here we consider how we might be able to control a vehicle so that it moves from one point to another. Obviously we won't use an open-loop controller because it is hard to verify that works as expected without any feedback. Thus we shall investigate a feedback system in order to find constraints that will determine the controller configuration of our design. Consider the situation shown in figure 14, with an arbitrary position and orientation of the robot and a predefined goal position and orientation. The actual pose error vector given in the robot reference frame $\{X_R, Y_R\}$ is $e =^R [x, y, \theta]^T$ with x, y, θ being the goal coordinates of the robot.
The task is to find a control matrix, K with

$$K = \begin{bmatrix} k_1 1 & k_1 2 & k_1 3 \\ k_2 1 & k_2 2 & k_2 3 \end{bmatrix} with\ k_{ij} = k(e, t) \tag{8}$$

such that the control of $v(t)$ and $\omega(t)$

$$\begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} = K \cdot e = K \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_R \tag{9}$$

drives the error towards zero

$$\lim_{t \to \infty} e(t) = 0 \tag{10}$$

We can describe the kinematics of our vehicle in the inertial frame $\{X_I, Y_I, \theta\}$ as

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{11}$$

To make our study of the system easier, we change the coordinates to polar form. Thus, we have

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -\cos\alpha & 0 \\ \frac{\sin\alpha}{\rho} & 1 \\ \frac{-\sin\alpha}{\rho} & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{12}$$

where the variables in the equation are defined in the figure 14 and $\alpha \in I_1$ where $I_1 = \left(-\frac{\pi}{2}, \frac{\pi}{2}\right]$. For $\alpha \in I_2$ where $I_2 = \left(-\pi, -\frac{\pi}{2}\right] \cup \left(\frac{\pi}{2}, \pi\right]$, our controller has a different configuration given below. First we set $v = -v$. Then we have,

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} \cos\alpha & 0 \\ -\frac{\sin\alpha}{\rho} & 1 \\ \frac{\sin\alpha}{\rho} & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{13}$$

7.2.1 A few remarks on the kinematics model in polar coordinate

- The coordinate transformation is not defined at $x = y = 0$; as in such a point the determinant of the Jacobian matrix of the transformation is not defined, that is unbounded.
- For $\alpha \in I_1$ the forward direction of the robot points toward the goal, $\alpha \in I_2$ for it is the reverse direction.
- By properly defining the forward direction of the robot at its initial configuration, it is always possible to have at $\alpha \in I_1$ *at* $t = 0$. However, this does not mean that α remains in I_1 for all time, t. Hence, to avoid that the robot changes direction during approaching the goal, it is necessary to determine, if possible, the controller in such a way that $\alpha \in I_1$ for all t , whenever $\alpha(0) \in I_1$. The same applies for the reverse direction.

We can investigate applying a linear control law to our system. Consider the control equations,

$$v = k_\rho \rho \tag{14}$$

$$\omega = k_\alpha \alpha + k_\beta \beta \tag{15}$$

We get the closed loop system given as

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -k_\rho \rho \cos\alpha \\ k_\rho \sin\alpha - k_\alpha \alpha - k_\beta \beta \\ -k_\rho \sin\alpha \end{bmatrix} \tag{16}$$

The system does not have any singularity at $\rho = 0$ and has a unique equilibrium point at $(\rho, \alpha, \beta) = (0, 0, 0)$. Thus it will drive the robot to this point, which is the goal position.

- In the Cartesian coordinate system the control law [equation(15)] leads to equations which are not defined at $x = y = 0$.
- Be aware of the fact that the angles α and β are restricted to the range $(-\pi, \pi)$.
- Observe that the control signal has always a constant sign, that is, it is positive whenever $\alpha \in I_1$ and it is always negative otherwise. This implies that the robot performs its parking manoeuvre always in a single direction and without reversing its motion.

There are further constraints on the system in order to ensure stability. This can be derived from our closed loop system. So our control system will be asymptotically/exponentially stable if

$$k_\rho > 0;\ k_\beta < 0;\ k_\alpha - k_\rho > 0 \tag{17}$$

Moreover, we can define a strong constraint that ensures that $\alpha \in I_1$ for all t, whenever $\alpha(0) \in I_1$ and $\alpha \in I_2$ for all t , whenever $\alpha(0) \in I_2$ respectively. This has been proven to work in practice.

$$k_\rho > 0;\ k_\beta < 0;\ k_\alpha + \frac{5}{3}k_\beta - \frac{2}{\pi}k_\rho \tag{18}$$

Thus we can see that the kinematics and control of the robot can be model completely using mathematically tools. This means that we can use tools available to use such as openModellica, Matlab and LabView to analyse these aspects of the mechanatronic system. These constraints when analysed will guide the robot's decision in setting the control configuration based on the values of α, the available trajectory to goal, etc.

7.3 Perception

In this section we will consider the mechanisms used by the vehicle to perceive its environment. The robot takes various measurements from sensors and extracts information from this data. There are various strategies for achieving perception so we will go through a few. There are two types of sensors used. These are the proprioceptive sensor which is used to collective information about the vehicles internal state , e.g. its speed and the exteroceptive sensor used to take measurements of the external environment, e.g. distance measurements. Sensors can also be classified as active or passive. Passive sensors measure ambient information coming in from the environment such as temperature while active sensors emit energy into the environment and then measure the surrounding's reaction to this energy, e.g laser range sensor. For this reason, active sensors suffer from interference. Table 1 can be used to compare the various available sensors for mobile robots. There are a few characteristics desirable in choosing an ideal sensor. This characteristics form the non-functional requirements for our sensors. Thus we want high dynamic range, high resolution, linearity, high bandwidth, high sensitivity, low cross sensitivity, high precision and accuracy, and low error in measurement.

7.3.1 Characterising Errors

Sensors take measurements in the local frame of the robot. However, the motion of our robot introduces errors in sensor readings. Thus we need to investigate and characterise these errors.

Blurring of systematic and random errors Active ranging sensors tend to have failure modes that are triggered largely by specific relative positions of the sensor and environment targets. For example, a sonar sensor will produce specular reflections, producing grossly inaccurate measurements of range, at specific angles to a smooth sheet-rock wall. During motion of the robot, such relative angles occur at stochastic intervals. This is especially true in a mobile robot outfitted with a ring of multiple sonar. The chances of one sonar entering this error mode during robot motion is high. From the perspective of the

moving robot, the sonar measurement error is a random error in this case. Yet, if the robot were to stop, becoming motionless, then a very different error modality is possible. If the robot's static position causes a particular sonar to fail in this manner, the sonar will fail consistently and will tend to return precisely the same (and incorrect!) reading time after time. Once the robot is motionless, the error appears to be systematic and of high precision. The fundamental mechanism at work here is the cross-sensitivity of mobile robot sensors to robot pose and robot-environment dynamics. The models for such cross-sensitivity are not, in an underlying sense, truly random. However, these physical interrelationships are rarely modeled and therefore, from the point of view of an incomplete model, the errors appear random during motion and systematic when the robot is at rest. The important point is to realize that, while systematic error and random error are well-defined in a controlled setting, the mobile robot can exhibit error characteristics that bridge the gap between deterministic and stochastic error mechanisms.

Multi-modal error distributions It is common to characterize the behaviour of a sensor's random error in terms of a probability distribution over various output values. In general, one knows very little about the causes of random error and therefore several simplifying assumptions are commonly used. For example, we can assume that the error is zero-mean, in that it symmetrically generates both positive and negative measurement error. We can go even further and assume that the probability density curve is Gaussian. It is important for now to recognize the fact that one frequently assumes symmetry as well as uni-modal distribution. This means that measuring the correct value is most probable, and any measurement that is further away from the correct value is less likely than any measurement that is closer to the correct value. These are strong assumptions that enable powerful mathematical principles to be applied to mobile robot problems, but it is important to realize how wrong these assumptions usually are.

Consider, for example, the sonar sensor once again. When ranging an object that reflects the sound signal well, the sonar will exhibit high accuracy, and will induce random error based on noise, for example, in the timing circuitry. This portion of its sensor behaviour will exhibit error characteristics that are fairly symmetric and uni-modal. However, when the sonar sensor is moving through an environment and is sometimes faced with materials that cause coherent reflection rather than returning the sound signal to the sonar sensor, then the sonar will grossly overestimate the distance to the object. In such cases, the error will be biased toward positive measurement error and will be far from the correct value. The error is not strictly systematic, and so we are left modelling it as a probability distribution of random error. So the sonar sensor has two separate types of operational modes, one in which the signal does return and some random error is possible, and the second in which the signal returns after a multipath reflection, and gross overestimation error occurs. The probability distribution could easily be at least bimodal in this case, and since overestimation is more common than underestimation it will also be asymmetric.

Table 1: Classification of sensors used in mobile robot application

General Classification	**Sensor System**	**PC or EC**	**Active or Passive**
Tactile sensors (detection of physical contact or closeness; security switches)	Contact switches, bumpers Optical barriers Non-contact proximity sensors	EC EC EC	P A A
Wheel/motor sensors (wheel/motor speed and position)	Brush encoders Potentiometers Synchros, resolvers Optical encoders Magnetic encoders Inductive encoders Capacitive encoders	PC PC PC PC PC PC PC	P P A A A A A
Heading sensors (orientation of the robot in relation to a fixed reference frame)	Compass Gyroscopes Inclinometers	EC PC EC	P P A/P
Ground-based beacons (localization in a fixed reference frame)	GPS Active optical or RF beacons Active ultrasonic beacons Reflective beacons	EC EC EC EC	A A A A
Active ranging (reflectivity, time-of-flight, and geometric triangulation)	Reflectivity sensors Ultrasonic sensor Laser rangefinder Optical triangulation (1D) Structured light (2D)	EC EC EC EC EC	A A A A A
Motion/speed sensors (speed relative to fixed or moving objects)	Doppler radar Doppler sound	EC EC	A A

EC-Exteroceptive, PC-Proprioceptive, A-Active, P-Passive

7.3.2 Our Sensor Selection

We now employ the tools described for classifying sensors to select sensors for our application. We choose to use laser ranger as our extroceptive sensor to gather information about our environment. We chose this because unlike the sonar, a laser sensor is less susceptible to distortion due to diffraction at corners. Also, we are assuming our environment is not made of glass so a laser ranger will work well and for a longer range than a sonar sensor. Moreover, data from a laser ranger can be processed easier than that from a vision based sensor. Thus a laser sensor can be seen as a good compromise between easy of use, interference properties and range.
For our proprioceptive sensor, we will use an absolute optical wheel encoder as this is cheap and readily available. The absolute encoder gives only speed of the wheel and not the direction. As opposed to using the incremental encoders and gyroscopes, it presents only quantisation errors that are follow a Gaussian distribution due primary to its low resolution. The other sensors have more complicated errors such as accumulated measurement bias over time. This means that the absolute wheel encoder is simpler to implement but it is useful enough to give us the level of detail we desire. Nevertheless, we may find that in practice that we will need to include a compass to give us absolute bearing information to correct for any errors in our sensors.

7.4 Vehicle Localisation

In this section, we are concerned about how we can estimate the location of the robot at any given time. This is one of the most difficult aspects of navigation but it has also received the most attention in research. There are two extremes to approach this problem. One is to avoid the problem altogether and hard code a way-point for the robot to follow in the environment. Here, the problems are numerous. First the robot is now operating in an open loop and does not take any feedback from the environment to confirm its position. Also, perfect knowledge of the environment is assumed. Obviously this approach fails in dynamic environment where systems are required to be robust and self-adaptive. Also, this is not a scalable solution. The second approach is to combine readings from the sensors taking feedback from the environment and using these to estimate the current location. We will follow the second route and investigate the numerous approaches. It is important to understand the challenges we face in this approach which are sensor noise and aliasing. Sensor noise may give an incorrect reading which can propagate to estimating an incorrect location. Thus an understanding of possible sensor noise is very important. For our chosen laser ranger sensor, we know that errors may be random in nature. Other time, it may be because the laser hit a glass-like material and does not reflect properly. There is also the issue of aliasing which really says, "based on the sensor readings, how do we distinguish between two locations?" There is also noise due to effectors. As the robot moves, the encoders may not properly measure its motion. This may be because the environment is not properly modelled and factors such as the slope of the floor, the robot may slip and a human may push the robot may not be taken into account. Here are some other causes of errors that may occur:

- Limited resolution during integration (time increments, measurement resolution, etc.);
- Misalignment of the wheels (deterministic);
- Uncertainty in the wheel diameter and in particular unequal wheel diameter (deterministic);
- Variation in the contact point of the wheel;
- Unequal floor contact (slipping, non-planar surface, etc.).

Some of the errors might be deterministic (systematic), thus they can be eliminated by proper calibration of the system. However, there are still a number of non-deterministic (random) errors which remain, leading to uncertainties in position estimation over time. From a geometric point of view, one can classify the errors into three types:

1. Range error: integrated path length (distance) of the robot's movement sum of the wheel movements
2. Turn error: similar to range error, but for turns difference of the wheel motions
3. Drift error: difference in the error of the wheels leads to an error in the robot's angular orientation

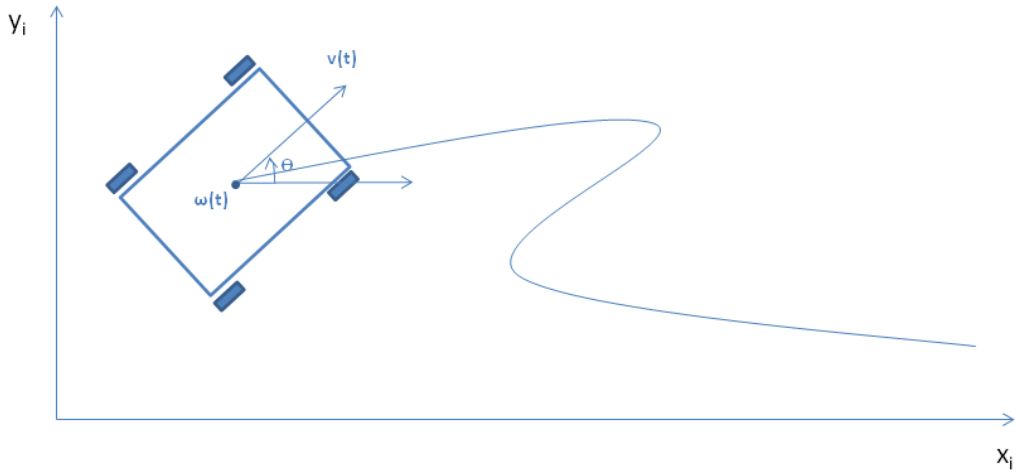

Figure 15: Movement of Differential Drive Vehicle

An error model for odometric position estimation Generally the pose (position) of a robot is represented by the vector

$$p = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{19}$$

For a differential-drive robot the position can be estimated starting from a known position by integrating the movement (summing the incremental travel distances). For a discrete system with a fixed sampling Δt interval the incremental travel distances$(\Delta x; \Delta y; \Delta\theta)$ are

$$\Delta x = \Delta s \cos\left(\theta + \frac{\Delta\theta}{2}\right) \tag{20}$$

$$\Delta y = \Delta s \sin\left(\theta + \frac{\Delta\theta}{2}\right) \tag{21}$$

$$\Delta\theta = \frac{\Delta s_r - \Delta s_l}{b} \tag{22}$$

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2} \tag{23}$$

where $(\Delta x; \Delta y; \Delta\theta)$ = path travelled in the last sampling interval;
$\Delta s_r; \Delta s_l$= travelled distances for the right and left wheel respectively; b = distance between the left and right wheels. Thus for updated position p' we get,

$$p' = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{\Delta s_r + \Delta s_l}{2} \cos\left(\theta + \frac{\Delta s_r + \Delta s_l}{2b}\right) \\ \frac{\Delta s_r + \Delta s_l}{2} \sin\left(\theta + \frac{\Delta s_r + \Delta s_l}{2b}\right) \\ \frac{\Delta s_r - \Delta s_l}{b} \end{bmatrix} \tag{24}$$

Odometric position updates can give only a very rough estimate of the actual position. Owing to integration errors of the uncertainties of p and the motion errors during the incremental motion $(\Delta s_r; \Delta s_l)$ the position error based on odometry integration grows with time. In the next step we will establish an error model for the integrated position p' to obtain the covariance matrix $\Sigma_{p'}$ of the odometric position estimate. To do so, we assume that at the starting point the initial covariance matrix Σ_p is known. For the motion increment $(\Delta s_r; \Delta s_l)$ we assume the following covariance matrix Σ_Δ:

$$\Sigma_p = covar(\Delta s_r, \Delta s_l) = \begin{bmatrix} k_r|\Delta s_r| & 0 \\ 0 & k_l|\Delta s_l| \end{bmatrix} \tag{25}$$

where Δs_r and Δs_l are the distances travelled by each wheel, k_r and k_l, are error constants representing the non-deterministic parameters of the motor drive and the wheel-floor interaction. As you can see, in equation(25) we made the following assumptions:

- The two errors of the individually driven wheels are independent;
- The variance of the errors (left and right wheels) are proportional to the absolute value of the travelled distances.

If we assume that p and $\Delta_{rl} = (\Delta s_r, \Delta s_l)$ are uncorrelated and the derivation of f [equation (24)] is reasonably approximated by the first-order Taylor expansion (linearisation), we conclude, using the error propagation law that,

$$\Sigma_{p'} = \nabla_p \cdot \Sigma_p \cdot \nabla_p f^T + \nabla_{\Delta_{rl}} f \cdot \Sigma_\Delta \cdot \nabla_{\Delta_{rl}} f^T \tag{26}$$

Once the error model has been established, the error parameters must be specified. One can compensate for deterministic errors properly calibrating the robot. However the error parameters specifying the non-deterministic errors can only be quantified by statistical (repetitive) measurements.

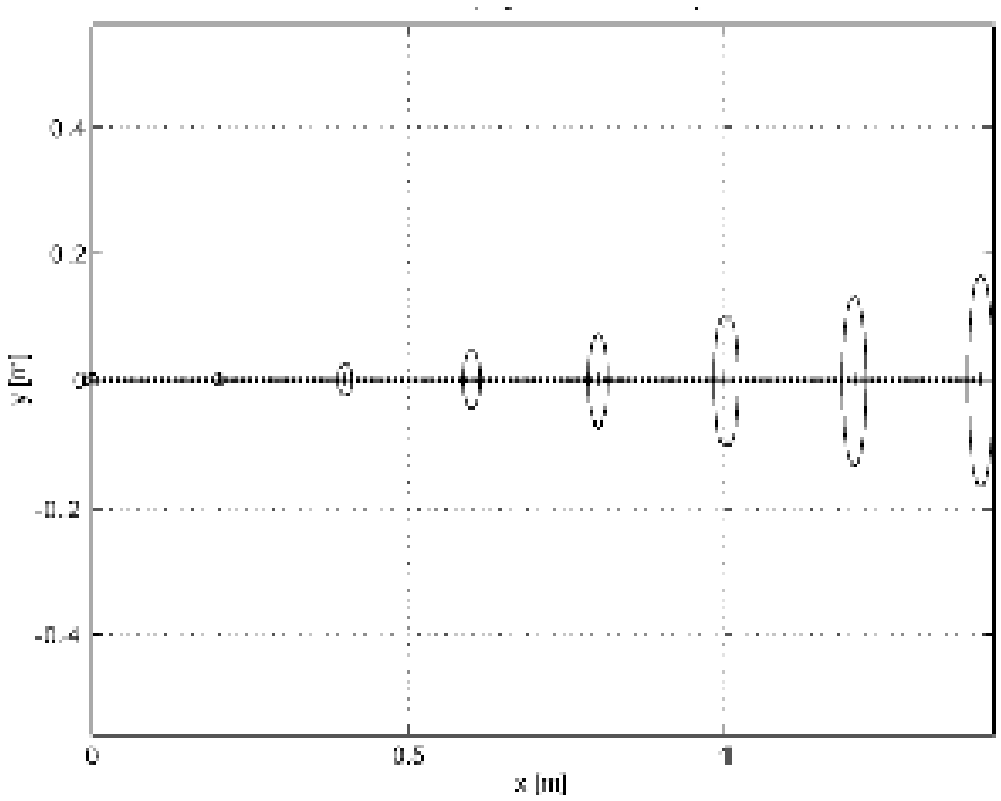

Figure 16: Error Propagation in Odometry

7.4.1 Belief Representation

In order to select from various localisation systems, its important we consider the belief representation for our map- based localisation. One can either subscribe to the single hypothesis or multiple hypothesis belief. In the single hypothesis, the robot's position is expressed as a single unique point. This removes all ambiguity in position which in turn facilitates decision-making at the cognitive level. However, the disadvantage is that motion inevitable produces uncertainty in position due to sensor and effector noise. Thus, forcing a position update to always generate a single hypothesis of position is often impossible.
On the other hand, multiple hypothesis allows for more than one hypothesis on the robot's location to account for uncertainty. The problem that arises in using unique multiple hypothesis is that it may become computationally expensive. Consider for instance a world in which there are on average N possible next positions when the robot moves arbitrarily, the number of possible belief states grows exponentially as $O(e^N)$. Even more problematic is the challenge of decision making as different beliefs may produce different trajectories. The way we will mitigate these challenges is to use a single belief along with a

representation of its uncertainty(this is called the error covariance). Thus the actual value is within the range expressed by error covariance. We thus can make decisions based on the mean which has the advantages of a single belief and at the same time, we can take into account the uncertainty of our belief which leverages the multiple hypothesis approach. We shall show an implementation of this approach as it pertains to our hybrid system but first we need to discuss how we can include the environment in modelling localisation.

7.4.2 Map Representation

The problem of representing the environment in which the robot moves tallies with the problem of representing the robot's possible position or positions. Decisions made regarding the environmental representation can have impact on the choices available for robot position representation. Often the fidelity of the position representation is bounded by the fidelity of the map.

Three fundamental relationships must be understood when choosing a particular map representation

1. The precision of the map must appropriately match the precision with which the robot needs to achieve its goals.

2. The precision of the map and the type of features represented must match the precision and data types returned by the robot's sensors.

3. The complexity of the map representation has direct impact on the computational complexity of reasoning about mapping, localization, and navigation.

Now the main aim of map representation is in properly estimating the robot's location. We may ask whether it is possible to abstract the map representation in order to optimise for memory and computational power. This is the so-called problem of decomposition.

7.4.3 Decomposition Strategies

1. Exact Cell Decomposition: Here we simplify the environment by representing the map as a continuous entity with a set of infinite lines that approximate real-world environmental lines based on a 2D slice of the surrounding. This is basically a filter that removes all non-straight data and extends line segments into infinite lines that require a few parameters.

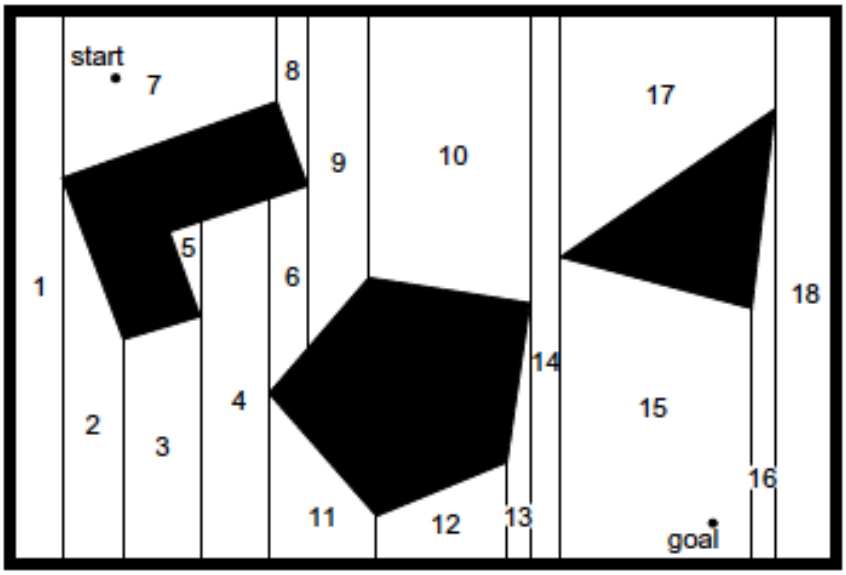

Figure 17: Exact Cell Decomposition

2. Fixed Cell Decomposition: Here we tessellate the environment transforming the continuous environment into a discrete approximation. The very popular variation of this is the occupancy grid in which cells are either free or occupied.

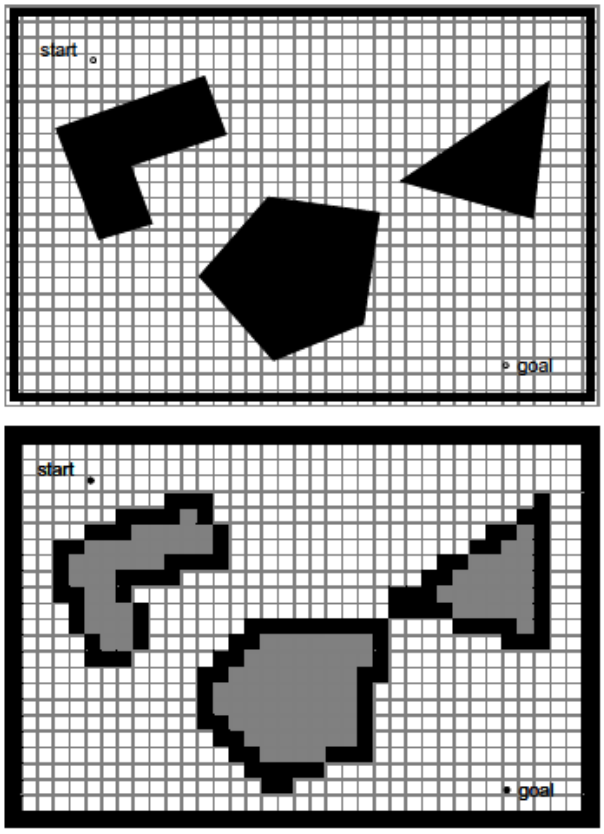

Figure 18: Fixed Cell Decomposition

We will use the occupancy grid with fixed grid size for our map representation. The main reason is that it is easy to implement robust path planning algorithms with a discrete map representation. The fixed size of our cell will be in the order of our laser ranger sensor uncertainty. The disadvantage however is that as the environment increases, more memory will be needed. Thus we must set aside memory for every cell in the matrix.

7.4.4 Probability Based Localisation

At the heart of our localisation is the Kalman filter algorithm which is responsible for providing estimates location based on the sensor readings. The kalman filter is an optimal and efficient sensor fusion technique. It works by first taking in the odometric sensors information to predict the robots next location on the map based on what the sensors describe as the motion of the robot over a certain time-step. This motion increases the uncertainty in the robot's location. The laser range sensors are then used to provide information about the environment. The re-observed features from the environment are used to update the robot's predicted position. There are other algorithms that could be used such as the particle filter. The advantage of the Kalman filter is that it is computationally less demanding than the particle filter once errors can be represented as Gaussian distribution. This is essentially true in our case as discussed previously.

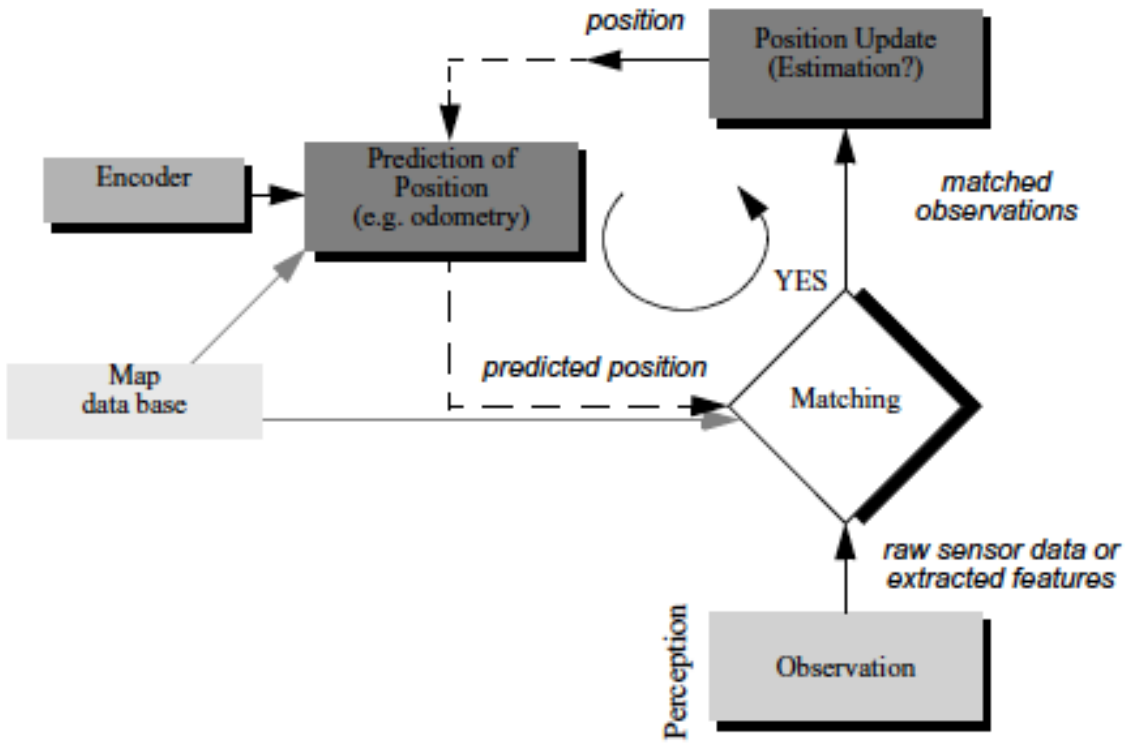

Figure 19: Kalman Filter Algorithm

7.5 Path Planning and Navigation (Cognitive Layer)

In the cognitive layer of an autonomous vehicle, navigation is at the core. Here the problem solved is that given a partial knowledge/map of an environment, navigation attempts to ensure that the vehicle reaches its goal as efficiently and as reliable as possible. There are two key parts of navigation. Given a map and goal location, path planning involves deriving a trajectory to the goal that will cause the robot to reach its destination once executed. Path planning is a strategic problem-solving competence that involves long-term decision making. On the other extreme is object avoidance which means modulating the trajectory to avoid obstacles.

7.5.1 The Challenge

To understand the challenge, let us consider a robot M at time i that has a map M_t and an initial belief state b_t. The robot's goal is to reach a position p while satisfying some temporal constraint: $location_{goal} = p;\ (g < n)$. Thus the robot must be at location p at or before time-step n. However, the robot can only sense its belief state not its actual physical location and therefore we map the goal of reaching location p to a belief state b_{goal}. A plan q will cause the robot's belief state to transition from b_t to b_{goal} if the plan is executed from a world state consistent with both b_t and M_t.

The problem is that the latter condition may not be met. It is very possible that the robot's position is not quite consistent with the belief state b_t and it is even more probable that M_t is either incomplete or incorrect. Moreover, the real-world environment may be dynamic. The planner's model regarding how M changes over time is usually imperfect.

Nevertheless in order to reach its goal, the robot must incorporate new information gained during plan execution. As time marches forward, the environment changes and the robot's sensors gather new information. This is precisely where reacting becomes relevant. In the best of cases, reacting will modulate robot behaviour locally in order to correct the planned-upon trajectory so that the robot still reaches the goal. At times, unanticipated new information will require changes to the robot's strategic plans, and so ideally the planner also incorporates new information as that new information is received.

A useful concept in discussion the architecture of the vehicle deals with the trade-off of a design decision and how it affects the system's ability to achieve a desired goal whenever a solution exists. This concept is called completeness. We will need to discuss the key aspects of planning and reacting as they apply to our mobile vehicle and how design decisions impact the potential completeness of the overall system.

7.5.2 Path Planning

Path planning techniques especially for differential drive systems are well studied and researched. We take inspiration from techniques invented originally for industrial manipulators. These techniques for industrial robot take into account both the dynamics and kinematics of the robot's motion.

Configuration Space Path planning is done in a representation called the configuration space. Here the degrees of freedom k are represents with k real values, $q_1, \ldots, q_k$. In our case we have k = 2 with $q_1 = v$; $q_2 = \omega$. Alternatively we can also have k = 3, with $q_1 = x$; $q_2 = y$; $q_3 = \theta$. The goal of path planning now is to find a path in the physical space such that all obstacles are avoided. This can be conveniently represented as the freespace F which is a subset of the configuration space C such that robot doesn't bump into obstacles represented by O. Thus F = C - O. Due to the non-holonomic contraints on our system, there may be further limitations to the free space of the robot's velocity. We may however assume in our case that the constraints are holonomic and since a holonomic path can be mimicked if the rotational position of our robot is not critical.

Path Planning Overview The robot's environmental representation can range from a continuous to decomposition-based geometric map. The first step of any path planning system is to transform continuous environment into a discrete map suitable for the chosen algorithm. There are three strategies idenfiable. These are:

1. Road Map: Identify a set of routes within the free space. These could either be the visiblity graph or the voronoi diagram.
2. Cell decomposition: discriminate between free and occupied cells. A common variation is the occupancy grid.
3. Potential field: impose a mathematical function over the space.

For our purposes, we would use cell decomposition but of its wide range of compatibility with many path planning algorithms.

Cell Decomposition Path Planning The basic cell decomposition algorithm can be summarized as follows:

- Divide the map M into simple connected regions called cells.
- Determine which free cells are adjacent and construct a "connectivity graph".
- Find the cells in which the initial and goal configurations lie and search for a path in the graph linking the two cells.
- From the sequence of cells found with an appropriate algorithm, compute a path within each cell , e.g. a path passing through the mid point of the cell.

We can implement a lossless placement of boundaries for the cells in which cells are places as a function of the structure in the map. A more practical implementation is an approximate cell decomposition where the result is an approximate of the actual map as seen in section(7.4). There are various algorithms that can be used for cell decomposition path planning. These include NF1 grass-fire which is essentially a breath-first search variant. Any graph search algorithm with the right heuristic can be used.

7.5.3 Obstacle Avoidance

Local obstable avoidance is concerned about changing/modifying the trajectory of a robot as informed by its sensors during motion. There exists several algorithms for this purpose. Some work together with a global path planner and some are very sophisticated taking into accounts the robot's kinematics and dynamics. The simplest algorithm is the bug. Here the approach is to follow a contour of the obstable around it then depart from the point with the shortest distance towards the goal. Other algorithms include bug2, Vector field histogram, bubble band technique and dynamic window. See [7] for more information on these algorithms.

Adding dynamic constraint A new kind of space representation was proposed by Minguez, Montano and Khatib [3] in an attempt to address the lack of dynamic models in most of the obstacle avoidance approaches. The technique is applicable to configuration space and workspace representation. It transforms obstacles into distances that depend of the braking constraints and sampling time of the underlying obstacle avoidance method. In our implementation, we will make the simple assumption that the robot is moving slow enough that the braking constraints are essentially negligible. Thus we will not be considering dynamic constraints any further.

7.5.4 Focussed D* Algorithm

We have considered the various algorithms for path planning and obstacle avoidance. The problem is that there isn't a cogent method of combining both the planner and the obstacle avoidance. A novel algorithm exists that solves this problem. Dynamic A* search also known as D* combines path planning and obstacle avoidance by planning optimal traverse in real-time whenever new information about the environment is found. In this way, it incrementally repairs the robot's path when the robot discovers unexpected obstacles that were not present in its map. The intuition behind D* is that similar to A* search, we start from point S, in a partial map of the environment and construct an optimal path sequence {G,S} from the goal G such that {G,S} presents the lowest cost generated by some optimal heuristic. The beauty of this heuristic is that it is arbitrary in that it could take into account the kinematic constraints of the chassis, the state of the processor and so on. As the robot moves, it may observe obstacles in its path or not. Either way, the presence of the obstacle may mean that the path currently taken is suboptimal. Hence, D* recalculates an optimal path in real-time by assigning the obstacle with a high cost and propagating path cost changes. We will adopt the focussed D* which is a more complete version of the algorithm [9]. With the focussed D* algorithm which uses a heuristic that only considers optimal cost from the robot's current location, this operation can be done very quickly in real-time and hence the robot does not need to stop to rethink. One reason for this is that sensors will detect obstacles local to the robot. Here we are assuming the obstacle is small. Thus only local fixes to the path are needed moreover as the robot moves generally closer to the goal, the path length to recalculate becomes smaller. Extensive proof of the optimality, soundness and completeness of this algorithm can be found in [8].

7.6 Navigation Architecture

Given techniques for path planning, obstacle avoidance, localisation and perceptual interpretation, how do we combine all of these into one complete robot system for real world application? We may wish to proceed to design a custom application specific software system. On the other hand, there is a need to study the possible navigation architectures and approach the problem in a principled manner partly for the scalability and portability of a solution that we desired as part of our non-functional requirements. This approach also ensures that we develop a long-term sophisticated solution which is useful especially in complex environment and dynamic environments. Thus we want all the qualities of a good software design which includes modularity where components such as sensors can be replaced by a different type whilst still ensuring that we do not have to redesign the entire system for compatibility.

We also desire control localisation. By localizing each functionality (path planning, path execution, etc.) to a specific unit in the architecture, we enable individual testing as well as principled strategy for control composition. For example we can localise all software involved in obstacle avoidance. At the other extreme, high-level planning and task-based decision-making software can be localised enabling exhaustive testing in simulation and thus verification without even direct connection to the physical system. A final advantage of localization of control is associated with learning. It enables specific learning algorithms to be applied to just one aspect of the robot's system. Such targeted learning is likely to be the first

strategy that yields successful integration of learning and traditional mobile robotics as it presents the simplest case for learning.

Focussing solely on navigation competence, the two types of decomposition that finds practical relevance for the robot's software are temporal and control decomposition.

7.6.1 Techniques for Decomposition

Decomposition identifies the axes along which we can justify discrimination of robot software into distinct modules. Temporal decomposition distinguishes between real-time and non real-time demands on mobile robot's operation. Control decomposition identifies the way in which various control outputs within the mobile robot architecture combine to yield its physical actions.

Temporal Decomposition In figure 20, we can see the generic temporal decomposition for navigation. The real-time processes are at the bottom of the stack with the highest category being occupied by processes with no real-time demands. The lowest level captures functionality that must proceed with guaranteed fast cycle time, such as a 40 Hz bandwidth. In contrast, the quasi real-time layer may capture processes that require for instance .1 second response time, with large allowable worst-case individual cycle times. A tactical layer can represent decision making that affects the robot's immediate actions and is therefore subject to some temporal constraints while strategic or off-line layer represents decision that affects the robot's behaviour over a long period with few temporal constraints on the module's response time. Four general interrelated trends correlate with temporal decomposition. These are not set in stone.

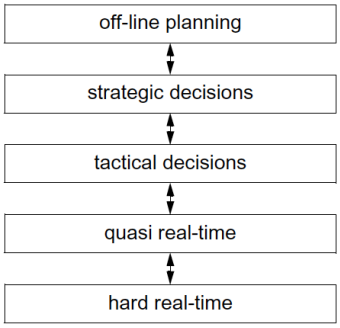

Figure 20: Generic Temporal Decomposition

Sensor Response Time A particular module's sensor response time can be defined as the amount of time between acquisition of a sensor-based event and a corresponding change in the output of the module. As one moves up the stack in the figure 20, the sensor response time increases. For the lowest level, the limitation is often from processing and sensor speeds. At the highest-level, sensor response can be limited by deliberate and slow decision making.

Temporal Depth This is a useful concept applying to the temporal window that affects the module's output both backwards and forward in time. Temporal horizon describes the amount of look ahead used by the module during the process of choosing an output. Temporal memory describes the historical time span of sensor input that is used by the module to determine the next output. Lowest modules tend to have very little temporal depths both ways, while deliberative processes of the highest-level modules make use of a large temporal memory and consider actions based on their long-term consequences making note of large temporal horizons.

Spatial Locality Together with temporal depth, the spatial impact of layers increases significantly as one moves from low-level modules to high-level modules. Lower level are more concerned with localised behaviour involving real-time wheel speed and orientation control. Higher levels project far into the future and may not be involved on local positioning.

Context Specificity A module may make decisions as a function of not only its immediate inputs but also as a function of the robot's context captured by other variables such as robot's interpretation of the environment. Lowest level modules tend to produce outputs as a result of immediate sensor inputs

using little context and are therefore relatively context insensitive. This is the reverse for higher level modules.

The cycle time, or bandwidth, of the modules changes by orders of magnitude between adjacent modules. Such dramatic differences are common in real navigation architectures, and so temporal decomposition tends to capture a significant axis of variation in a mobile robot's navigation architecture.

Control Decomposition Control decomposition divides based on the ways in which each module's output contributes to the overall robot control outputs. The basic principles of discrete systems representation and analysis is essential in presenting control decomposition. Consider the robot algorithm and the physical robot instantiation (i.e., the robot form and its environment) to be members of an overall system S whose connectivity we wish to examine. This overall system S is comprised of a set M of modules, each module m connected to other modules via inputs and outputs. The system is closed, meaning that the input of every module is the output of one or more modules in M. Each module has precisely one output and one or more inputs. The one output can be connected to any number of other modules inputs. We can specify a special module r which represents the physical robot. The output of r represents the complete perceptual output I to the robot comprising of the many discrete sensors and degree of freedom. The inputs of r are the complete action specification, O of the robot. From the viewpoint of the rest of the control system, the robot's sensor values are inputs and the robot's actions are outputs.

7.6.2 Case Studies: Adaptive Systems Architecture

We will consider two architectures that try to utilise the general control and temporal decomposition. We will not be concerned with architectures which have significant offline planning because this singular trait is not desirable in an adaptive system. Consider a case where a robot encounters an unknown obstacle. Surely an architecture with much emphasis on off-line planning will not be able to handle this situation nor work in a dynamic environment.

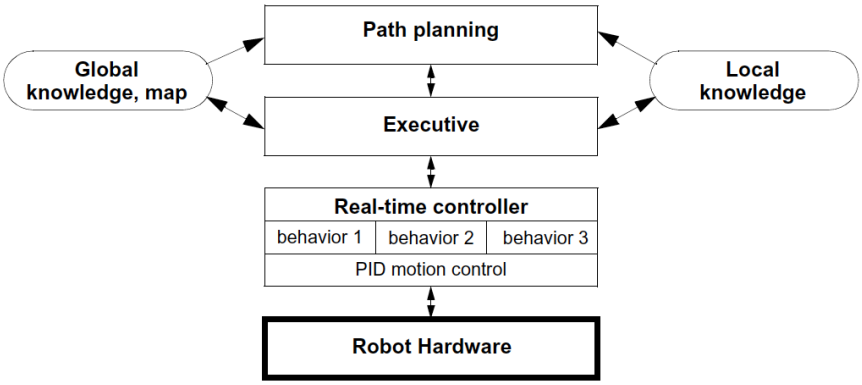

Figure 21: A three-tiered episodic planning architecture.

Episodic Planning This is perhaps the most popular method for robot navigation because it includes and processes new information in a computationally tractable way. The structure is three tiered as shown in figure 21. Planning is computationally expensive and therefore it is not done too frequently. A trigger that would initiate replanning could be when the robot detects a new obstacle. A common form of episodic technique is called deferred planning in which the robot re-plans and modifies its map after reaching its goal. A more sophisticated approach would be that when the lowest level behaviour can not make progress, the next upper level kicks in to take control. In this case, it is the **executive layer** which is responsible for activating and reactivating behaviours based on information it receives from the planner. It is also responsible for determining if a behaviour fails and re-initiating the planner. Moreover, it is common for this architecture to have both a global map and a temporary local map. Here

it is the executive's job to decide if new information from the local map should be integrated in the global knowledge base.

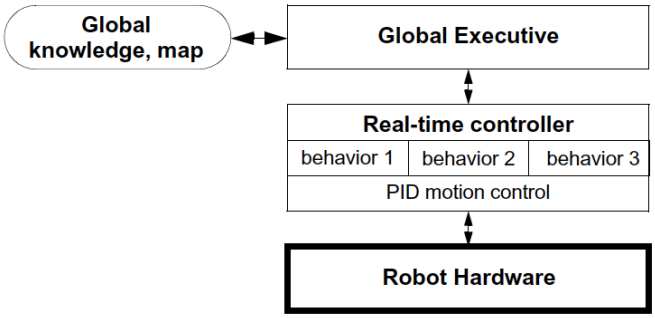

Figure 22: A two-tiered integrated planning and execution Architecture.

Integrated Planning and Execution In figure 22, we see an architecture that is basically two parts. It seems the planner may have disappeared but this system is really more advanced than the episodic case. The planner has become simply a small part of the executive nominal cycle of activities. The idea of speeding up planning such that its execution time is no longer significant may seem wishful. In fact this is what we achieve with our chosen D* Star algorithm. The advantage of this approach is that at every stage, the robot is guided by a global map. However, it may seem that this method is computationally demanding and will not be practical in very complex environments until processor speeds increases. This has yet to be a problem in real-life scenarios where this architecture has been implemented. The major success of this architecture is that the designer doesn't just consider all aspects of the robot and its environment task but must also consider the state of the processor and memory technology. In general, it is expected that there will be more research in robot architecture for years to come. With all forms of progress from robot sensor inventions to microprocessor speeds increases, there are likely to catalyse new revolutions in architecture as previously unimaginable tactics becomes realities.
Earlier in section 2.1, we used the term viewpoint to refer to a perspective. Hence, functional decomposition is essentially the same as functional viewpoint. We can also employ functional decomposition in arriving at a principled approach to navigation architecture. Nevertheless, both control and temporal decompositions are extremely useful for providing information on the hybrid system further than the functional decomposition discussed in section 2.1 can.

7.7 Context Investigation Discussion

From our investigation, we can see that different approaches are taken to modelling different aspects of the mechatronic system. The generic aspects of a mechatronic adaptive system in our case study are the major components that make up the navigation system such as the localisation and perception aspects. Navigation itself can be seen as only an aspect of an entire adaptive system. Borrowing the principles from [1] (chapter. 7), I propose there are five generic parts of every adaptive system.

1. The sequencer to generate behaviour in order to accomplish subtask. This could be a state machine.
2. The resource manager which allocates resources to behaviours. E.g it may select which sensor to be used for perception if there are various types to choose from (sonar, laser ranger, visual).
3. Cartographer to store the map data and the methods to access them.
4. A planner which interacts with humans and constructs mission plans.
5. The performance monitoring and problem solving agent which allows the agent to identify when tasks are not going as expected. This is a form of self-awareness.

Now the navigation system in our case study can be classified as the planner. Nevertheless, it would appear that the navigation system itself can serve as a representative of the entire system similar to a miniature world within a world. The reason is these five cores of an adaptive system can be found within the navigation. Thus, it is sufficient to say that whatever principles we extract from our case study which focuses on the navigation system can be extrapolated and applied to an entire adaptive system. Another

way to view this is to consider the navigation to be a simpler version of an adaptive system while a more complex system will consist of simple sub-systems.

A first step in modelling is to study theories behind the various parts of the system. These include the mechanical model, the control system, etc. The aim of this study is to derive the exact parameters and limitations that would guide the design process. For instance the mechanical model may impose strong conditions on the capabilities of the control system. These conditions could be dependent on the environment in which the robot is designed for and thus the environment as well must be studied as extensively as possible. With the understanding of the various parts in mind, we can then proceed to think of how best to combine these parts. This is the study of the system architecture. Before proceeding any further, I invite you to think of mechatronics as a synergy between the mechanical, electrical and software engineering. It is not merely the combination of these sub-domains but a separate field of study in itself (perhaps more advanced than any of its sub-domains) focusing on how these sub-domains can be merged together for a task. As such our context investigation is not subject specifically to any sub-domain and should not be split into the well established sub-domains. As can be observed, kinematics studies which has origin in mechanical engineering was combined synergistically with our control system (electronics engineering) to derive the equations for the control system. It is this interaction between the various sub-domains that defines Mechatronics. With this in mind we can reason about hybrid systems without strictness the syntax of a particular sub-discipline. Nevertheless practically, most engineers are specialised in only one sub domain. In this case, it appears that for synergy, all collaborating engineers must be knowledgeable to some extent of all areas of mechatronics and how design in one area may affect other areas. For very complex systems, very specialised task can be delegated to very specialised engineers by those who have considered the potential impacts of the design task on the overall system.

Proceeding on, the parts of a hybrid system can be grouped hierarchically into three three broad layers. These are the cognitive layer responsible for long term planning, the executive layer which contains the automaton used in managing the lower layer called the controller. The controller provides signals to the actuators and is responsive for smooth transmission between two continuous states e.g, smooth change in motor speed and direction. Detailed modelling of the system is done from the ground up starting at establishing a kinematic model of the system. This model provides information of the degrees of freedom the hybrid system has over itself. The environment or workspace in which the system will operate has to be studied extensively as well. This study will provide information of how the robot can use its degrees of freedom to carry-out its task in its environment. In very dynamic workspaces where all environmental factors can not be modelled or accounted for, designers must be able to impose hard constraints on the system to mitigate the effects of un-modelled conditions. For example, we imposed a strong condition on the control system of our vehicle for practical reasons where our controller may fail to work due to circumstances accounted for. The kinematic constraints and control system models can be developed mathematically to be simulated using tools such as OpenModellica, Matlab and LabView.

The next step in the modelling process is to combine the various well understood components by defining a suitable architecture for use. We proposed two types of architecture for adaptive systems. The episodic planning and the integrated planning with execution. While the episodic planner is very computationally agreeable, sophistication in autonomy will likely come from the integrated approach because it seamlessly combines many aspects of the design whilst achieving optimality. In fact, with algorithms such as focussed D* at the heart of the architecture, completeness, optimality and soundness properties are preserve at every point in time whilst taking into account other properties such as the memory design and the state of the processor. Thus better algorithms eliminate the problem of balancing global and local optimisation of resources such as power as would have been the case if we use a global path planning algorithm (such as the NF1) with a local couterpart like the Bug2.

Both steps in the modelling process require different approaches and tools. The first step which involves understanding the components in the three layers of the hierarchy, require modelling practices native to the theories from the sub domains which the component can be readily identified with. For example, modelling the physical system Nevertheless, there may also be interactions with other components from other sub domains. In this case, the true essence of mechatronics appears as the parameters and constraints on a layer are taken into consideration when designing components of the next upper layer as we did in our case studies

8 Modelling Non-functional Requirements

Non-functional requirements range from reliability and safety assurance to minimum power consumption. They can be described as constraints used to qualify but not modify the functionality of a system or

component. From our previous section, we saw that non-functional requirements such as optimality of path chosen can be embedded within the modelling of the functional requirements. Nevertheless, there are some non-functional requirements that should be explicitly modelled due to their importance so that they may be understood clearly. Such a requirement is the reliability or safety of the system.

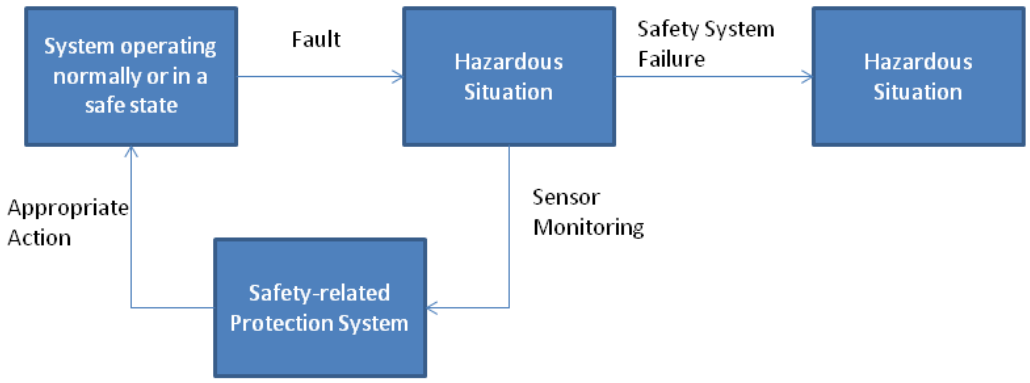

Figure 23: Safety Relationship

8.1 Modelling Safety of the system

In order to fully understand the modelling process, it is important to be clear about the meaning of certain key words. The relationship between these key words is shown in fig 23. These terms are:

- Accident: The actual event which causes harm.
- Hazard: A situation, such as a fault which has the potential to cause harm.
- Risk: A combination of the probability of a hazard occurring and the severity of the resulting accident.
- Level of Safety: The level at which the risk is at an acceptable level.

Now in real systems, absolute safety is not a valid concept in design. Nevertheless, a reasonably low risk level can be expected. The first step in the modelling process is to perform a hazard and risk analysis. The aim of these is to:

1. Identify all the hazards under all conceivable circumstances.
2. Determine the chain of events that leads to the hazards and the consequences of the hazard.
3. Determine the risk associated with each hazard such as the frequency and severity.

There are several techniques for performing risk analysis. Fault tree and event tree analysis [2] (Chapter 9) in particular is an established technique that follows a deductive reasoning approach to establish how a chain of events can be traced from a top level event. Nevertheless, it tends to be useful when failure follows discrete steps. In our vehicle however, this model will prove insufficient when considering instances where failure may happen in a continuous process theoretically. Let us explore this possibility.

Defining Hazards Hazards are defined initially in a generic way to avoid making any initial assumption about the system and environment. Some of these hazards are grouped into: (1)Collision with an object in path.(2)Toppling of robot.

Hazard analysis Applying the fault tree analysis is inadequate for several reasons given in [2]. These are:

1. It was hard to decide whether each level of the top-down deduction was to represent a temporal change or a more detailed description of the event above with no movement in time.
2. It is impossible to represent the physical reasons why hazards like toppling occurred using discrete events, but these reasons had to be understood to know how toppling is caused.

3. The tree would become nothing more than a disjointed representation of what is already known.

The reason for this established FTA method failing is that it was created to establish a sets of faults which in combination could give rise to a particular hazard. Hazards like toppling and collision can happen without a fault occurring in a dynamic environment simply due to the combination of circumstances. Moreover FTA was developed of control systems working in structured environment. What we want is a method that allows for safety reduction to a satisfactory level in a dynamic and possibly unstructured environment. Mathematical expressions can be used to model the way toppling and collision may occur. For collision, it is important to consider the speed at which an object is approaching the vehicle and vice-versa. For toppling, we need to know about the slope of the robot; the mechanical dynamics such as distribution and centre of mass; the external forces such as wind pressure which may be applied on the vehicle. There are two ways of ensuring safety.

1. Add an extra functionality to handle dangerous situations.

2. Place limitations on the environment to mitigate risk.

Clearly robust safety approaches will stem from the first method. Thus it is important to investigate the architectural model for this extra functionality.

8.2 Safety System Architecture

Hardware The main sensors are: the already existing optical laser sensor to detect obstacles and a two axis tilt sensor. The laser scans through 360 degrees and can give an indication of when an object has entered within a zone of caution in which a hazard is imminent. These sensors are connected to a "safety manager" processor.

software The safety manager needs to be aware of the events of a hazard in order to avoid an accident. It depends on the information to which it has access to and the amount of control it has over other modules of the system. It can only make decisions on the presence of a hazard if provided with enough information and can only carry out corrective measures if it can control the actuators. Thus the first question to ask is type and manner of communication between it and the other modules of the vehicle and not what the managers functionality would be like. [2] suggests a CANbus system which uses a broadcasting paradigm where all transmitted messages are sent to all nodes on the bus which decide locally whether or not the message is relevant to them. The aim is that the safety manager has access to all communication information about the system and not just that specifically meant for it. A communication protocol is needed which includes information on:

1. The hazard analysis, the results of which include the type of results the manager requires about the position, slope and speed of the system; the imminent action of the other modules and the state of the environment.

2. The type of information which needs to be passed on between modules

3. The fact that the safety manger must be aware of the status of all other modules so as to maintain safety in the event of a component failure.

The results of the hazard analysis has to form part of the input to the entire design process not just the software design of the manager. It is clear that the safety manager can not be the only safety critical part of the system since it depends on other components for information. Thus the architecture of our system must be redesigned to take into account these time-critical changes. The question here is "how exactly can the manager be integrated with the rest of the system?". In any case, an already existing case study given in [2] details the functionalities of the safety manager. These are:

1. Checking the health of other components and the health of the sensor.

2. Checking the local environment for intruding objects.

3. Checking a data-base for predefined hazards and using global positioning to see if they are in locality.

4. Receiving proposed action commands from the activities manager.

5. Consulting a rule-base of safety to see if proposed actions are safe.

6. In the event of the proposed action being unsafe, a human supervisor may stop the machine using independent hardware link or slow down the vehicle in case of collision.

Taking an example, if our vehicle detects an approaching object has reached a region in which a collision will most likely occur, the safety manager takes control of the controllers in order to slow down the system. Other functionalities such as the localisation and path-planning will till be functional. Only the navigation will be overridden temporarily till the risk of hazard is over.

Implementing safety Manager's Integration with System's architecture Let us define the structural components of our system as shown in figure 24. In this figure, we shown the structural connection of the key components along with the direction and labelling of the data flow between them. We also show the behaviour of the safety manager in figure 25) and its interaction with other components in case of a detected hazard in figure 26.

- Path Planner: This is essentially a D* algorithm that produces a sequence of states which describes the path to be taken.

- Mapping and Localisation: This module does the double job of producing the map and localising the robot's position within the map. Gmapping which is opensource can be used here.

- Perception: Here the optical sensors together with its driver is packaged. The output are raw data that the mapping unit processes.

- Motor Control System: This takes in high level instructions from either the path planner or safety manager and process this into continuous signals that that motor can understand. The path planner sends a way-point for the control to follow. The control's job is to attempt to follow this waypoint as best as it can. If it can't loses its way then it should request for the planner to send recalculate a new path based on current location. Here we are assuming that the control system will not fail to follow the safety manager's instructions because these will consist of simple and shorter duration instructions. Such an assumption has to be tested in practice or in simulations.

- Tilt sensors: These provide the information of the vehicles orientation in relation to the earth.

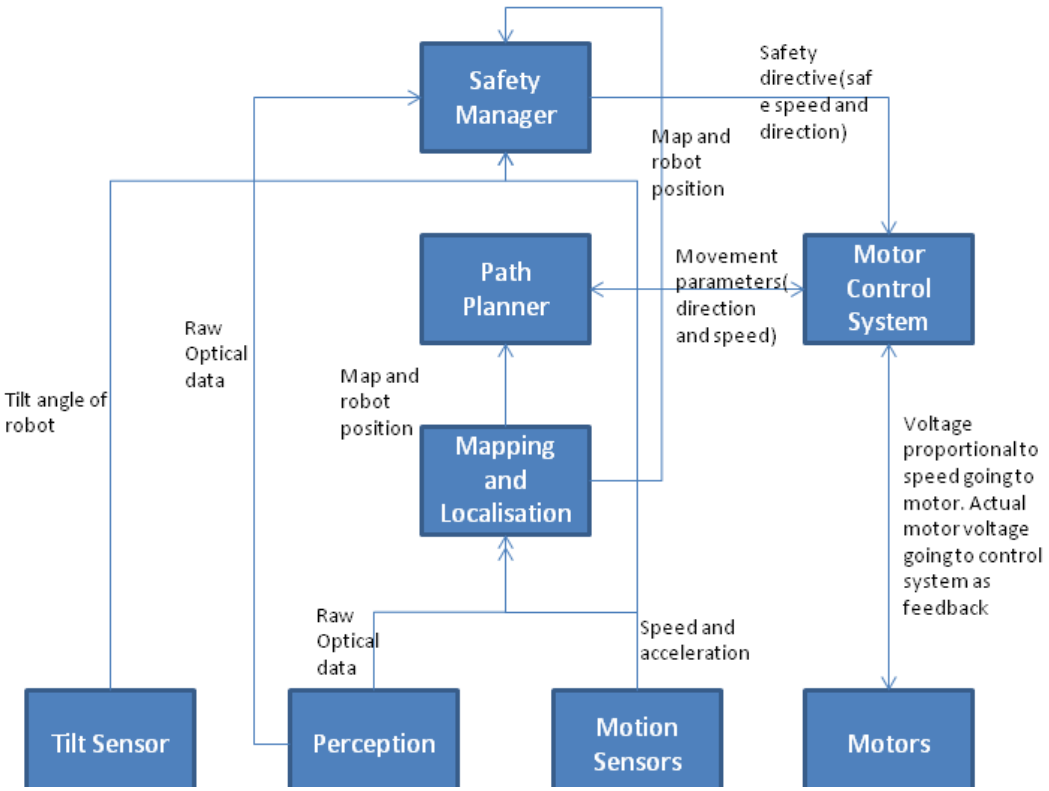

Figure 24: Structural Highlevel Redesign

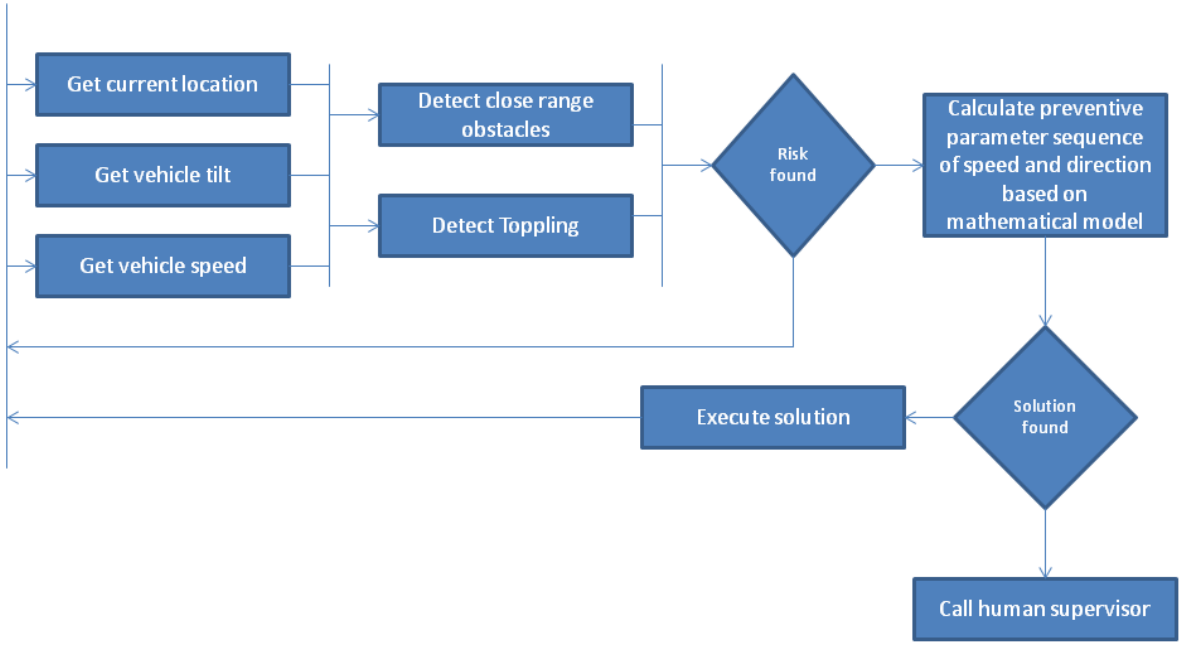

Figure 25: Safety Manager Behaviour

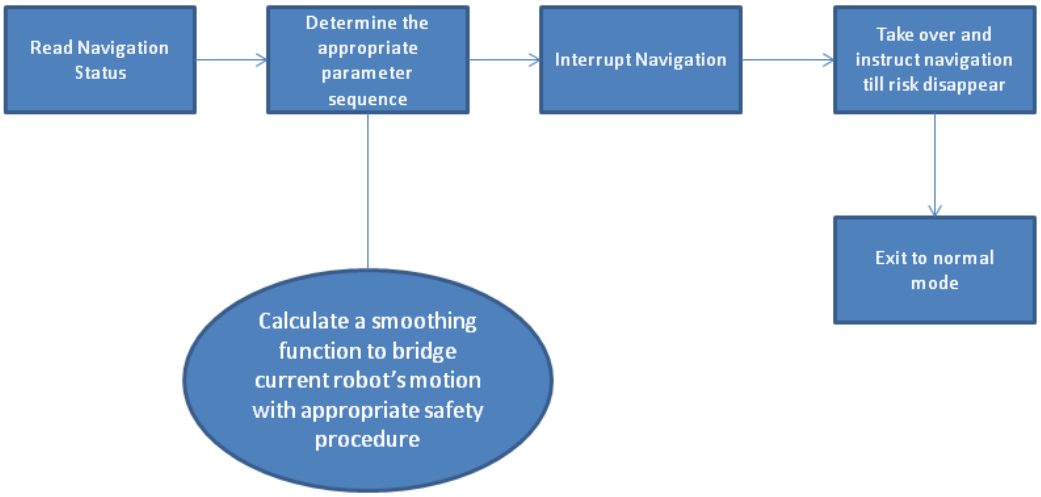

Figure 26: Behaviour showing safety manager interaction with other components to ensure safety

8.3 Discussion on Non-functional Requirement Modelling

In general however, non-functional requirements can either be embedded implicitly in the core functionality of the system's design or explicitly as we have shown in this section. The implicit design is useful for requirements that are simple enough to be embedded within the fabric of the core functionality. In our case for instance, the speed-limit requirements, optimal power consumption by motors by moving in the path with least cost, soundness and optimality of trajectory chosen by algorithm can all be modelled implicitly; safety requirements however can but should not. The reason being that safety is off a high priority and is most likely to involve all the components of the system, thus it is not a requirement that can be localised to one area of our hybrid system. Such a requirement should be modelled explicitly and treated as a separate entity from the core functionality. The interaction between safety module and the rest of the system must also be thoroughly investigated.

9 Conclusion

In this report, we have been able to investigate various principles involved in designing self-adaptive mechatronic systems. It is clear that lots of the innovation and design efforts in hybrid systems are in the software as opposed to the hardware. We were able to implement and analyse a case study of hybrid intelligent system which is the autonomous vehicle. We tackled the issues and/or challenges mentioned in [5] such as defining the design space of self-adaptive systems; defining the innovative processes for development, deployment and evolution of self-adaptive systems; and practical runtime verification methods for non-functional requirements for mechatronic systems. Nevertheless, there is still much work to be done in modelling more complex and robust systems than that which we have done in order to develop and verify modelling principle when dealing with higher levels for intelligence abstraction in order to deduce the general or universal principles in mechatronics intelligent systems design.

References

[1] Ronald C. Arkin. *Introduction to AI Robotics*. The MIT Press, 2000.

[2] D. Dawson S. Burge D. Braley D. Seward. *Mechatronics and the Design of Intelligent Machines and Systems*. Stanley Thornes, 2000.

[3] O. Khatib J. Mínguez L. Montano. Lausanne, 2002.

[4] J. Kramer and J. Magee. "Self-Managed Systems: an Architectural Challenge". In: *Future of Software Engineering, 2007. FOSE '07*. 2007, pp. 259–268. DOI: 10.1109/FOSE.2007.19.

[5] Rogerio de Lemos et al. "Software Engineering for Self-Adaptive Systems: A second Research Roadmap". In: *Software Engineering for Self-Adaptive Systems*. Ed. by Rogerio de Lemos et al. Dagstuhl Seminar Proceedings 10431. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2011. URL: http://drops.dagstuhl.de/opus/volltexte/2011/3156.

[6] Jeff Kramer Pedro Rodrigues Emil Lupu. "On the Role of Management in Self-Adaptive Systems". In: (2010).

[7] Illah R. Nourbarkhsh Roland Siegwart. *Introduction to Autonomous Mobile Vehicle*. The MIT Press, 2004.

[8] Anthony Stentz. "Optimal and Efficient Path Planning for Unknown and Dynamic Environments". In: *International Journal of Robotics and Automation* 10 (1993), pp. 89–100.

[9] Anthony Stentz. "The Focussed D* Algorithm for Real-Time Replanning". In: *In Proceedings of the International Joint Conference on Artificial Intelligence*. 1995, pp. 1652–1659.