Project Title

Bachelor's Technical Project Report
by

Vijeth Hebbar (150010046)

under the guidance of

Prof. Srikant Sukumar

Indian Institute of Technology, Bombay
Mumbai 400 076

Contents

Chapter 1

Abstract

A paper was published recently by Rihab Abdul Razak, Prof. Sukumar Srikant and Prof. Hoam Chung which designed an adaptive and decentralized control algorithm to obtain optimal sensing coverage using non-holonomic bots. A density function was assumed over the region to be covered, which could be viewed as a probability density of the phenomena to be sensed. The density function was an unknown but was assumed to be linearly parameterized with unknown parameter weights. A second order dynamical model of the non-holonomic bot was considered and an adaptive control law was designed to achieve optimal sensing coverage of the unknown environment.

In this work we try to implement the above designed controller to a non-holonomic bot Firebird V® developed by Nex Robotics®. There are numerous challenges in this implementation. We need to achieve a torque controller in absence of a current controller in the motor. Further, due to limited computational power on-board each bot care has to be taken in the computational intensive Voronoi partition generation.

These and other challenges have to and will be dealt with in designing an implementable control law to the bot.

Chapter 2

Voronoi Diagrams based Optimization

Voronoi Diagrams are extensively used to visualize as well as solve locational optimization problems. These include problems like,

- Where is the best place to stay in a neighbourhood if you want easy access to a shopping center, gym, school as well as the metro station?
- Where to stay if you want to avoid the noisy airport, garbage collection post, certain areas of neighbourhoods known for crime as well as some traffic plagued roads?
- What are the best places to place a mail box and the central post office based on the population distribution of a region?
- Or as the project addresses where to place certain sensors such that we obtain optimal coverage for sensing a certain environment?

We consider a region S in space $\mathbb{R}^n$ which contain some lower level manifolds defined in it. These manifolds may be points, lines, or higher order hyperspaces (or a combination thereof). Voronoi diagrams create tessellations in the region S by associating all locations in the space close[1] to these manifolds. 2.1 shows the partitions created in 2D space with points as the manifolds. Predominantly, there are two types of Voronoi diagrams - nearest point Voronoi diagrams and farthest point Voronoi diagrams.

[1]Can be defined in terms of Euclidian distance, Manhattan distance etc.

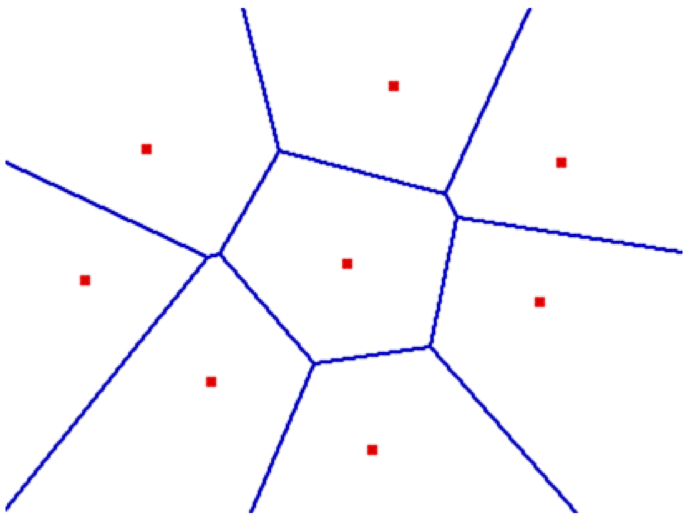


Figure 2.1: Nearest point Voronoi diagram

Let S be a space $\mathbb{R}^n$; s_i be a manifold in S and for a point p the distance from a manifold s_i is given by $d(p, s_i)$. Then the nearest point Voronoi partition for the i^{th} manifold is defined as in Eq. 2.1 and the furthest point Voronoi partition is defined as in Eq. 2.2,

$$V_i = \{p | d(p, s_i) \leq d(p, s_j);\ p \in S \backslash \bigcup_{1}^{n} s_i,\ i \neq j,\ j = 1, .., n\} \tag{2.1}$$

It is the set of all points around the manifold which are closest to the manifold. The 2.1 also shows a nearest point Voronoi diagram where any point in the partition is closest to the point at the center of the partition as compared to any other point manifolds in the space.

$$V_i = \{p | d(p, s_i) \geq d(p, s_j);\ p \in S \backslash \bigcup_{1}^{n} s_i,\ i \neq j,\ j = 1, .., n\} \tag{2.2}$$

It is the set of all points in the space S which is further from manifold than any other point.The Figure 2.2 also shows a furthest point Voronoi diagram.

We will next look at some problems that were stated at the starting of this section and look at the solutions provided for same by the usage of Voronoi partitions. We will look to find the largest empty circle in a set of points, smallest enclosing circle for a set as well as optimal distribution of facilities in a population distribution. Many more problems and their approaches are discussed in (4).

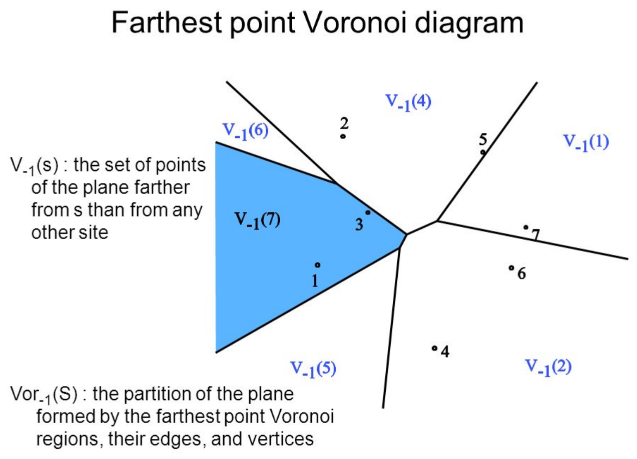

Farthest point Voronoi diagram
V-1(4)
V-1(6)
V-1(1)
V-1(s) : the set of points of the plane farther from s than from any other site
V-1(7)
2
5
3
7
6
1
4
V-1(5)
V-1(2)
Vor-1(S) : the partition of the plane formed by the farthest point Voronoi regions, their edges, and vertices

Figure 2.2: Furthest point Voronoi diagram

2.1 Largest Empty Circle

This problem seeks to find the the largest circular region inside a distribution of facilities/manifolds such that it does not include any facility. It is the furthest one can be from all facilities and still be inside the region defined by the set of outermost facilities. The same is seen in Figure 2.3. Mathematically the radius and center of this circle can be represented mathematically as in 2.3 and 2.4,

$$R = \max_{p \in S \setminus \bigcup_1^n s_i} \min_i \{d_E(p, s_i)\} \tag{2.3}$$

$$p_c = \arg \max_{p \in S \setminus \bigcup_1^n s_i} \min_i \{d_E(p, s_i)\} \tag{2.4}$$

Now in the nearest point Voronoi diagram it is easy to see that points on the line are at the maximum distance from any of the nearest points. Moving perpendicular to the line at any location brings us closer to at least to one of the points in the vicinity. Thus the center for largest empty circle must lie on the boundaries of Voronoi diagrams. Further, the bisectors of lines joining any point with any other two points are concurrent with the bisector of line joining the latter two points. [2]

If we move in any direction from such a point which involves intersection of three partition boundaries we will again move closer to at least one of the three points.

[2]Note that the point of intersection of the first two bisectors must be equidistant from all the three points.

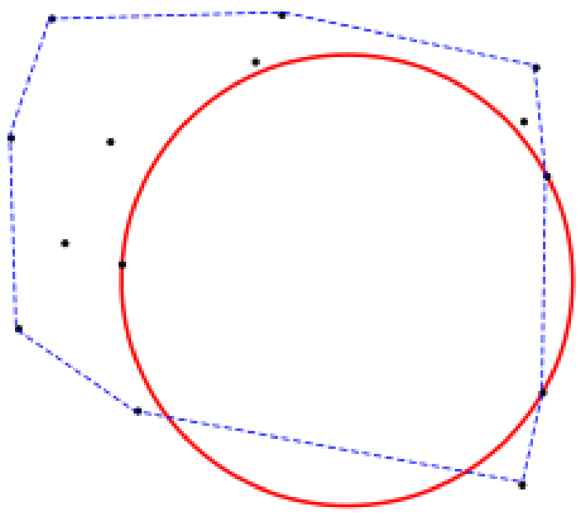


Figure 2.3: Largest Empty Circle (Wikimedia)

Thus it is intuitive to show that the center of the largest empty circle must lie at one of such points. All that remains now is to find the point (from the finite set of points) which has the maximum distance from the closest facility.

We assumed the distances in the 2.3 and 2.4 to be defined in a Euclidean manner. Changing the definition to Manhattan distance[3] or increasing the dimensions of both the space and the manifold gives analogous results.

2.2 Smallest Enclosing circle

We try to find the center as well as the radius of the smallest circle that can enclose all the points/facilities. This problem is analogous to finding the best house in a neighbourhood such that the distance to furthest facility from the house in minimized. Mathematically the center and radius are given as.

$$R = \min_{p \in S \setminus \bigcup_1^n s_i} \max_i \{d_E(p, s_i)\} \tag{2.5}$$

$$p_c = \arg \min_{p \in S \setminus \bigcup_1^n s_i} \max_i \{d_E(p, s_i)\} \tag{2.6}$$

[3]To take the furthest distance from an eatery in a city map

Now analogous to the analysis in the previous problem we will look at the furthest point Voronoi diagram. Points on the boundary of the partitions here are at a maximum distance from both the points they attempt to separate. Moving perpendicular to this line will bring us to a point further away from one point than the other. Again the point of intersection of three boundaries will be at the maximum distance from all the three points simultaneously. Thus we have achieved the $\max_i\{d_E(p, s_i)\}$ part of equation 2.5. Further any circle drawn through this point will intersect three points and will at the same time include all the other points in the space S.

While a formal proof isn't shown here, we can see from Equation 2.2 that such a point is equidistant from the three manifolds/facilities in consideration and further all other points in space are closer to the point than them. Let's take the set of all such circles as C.

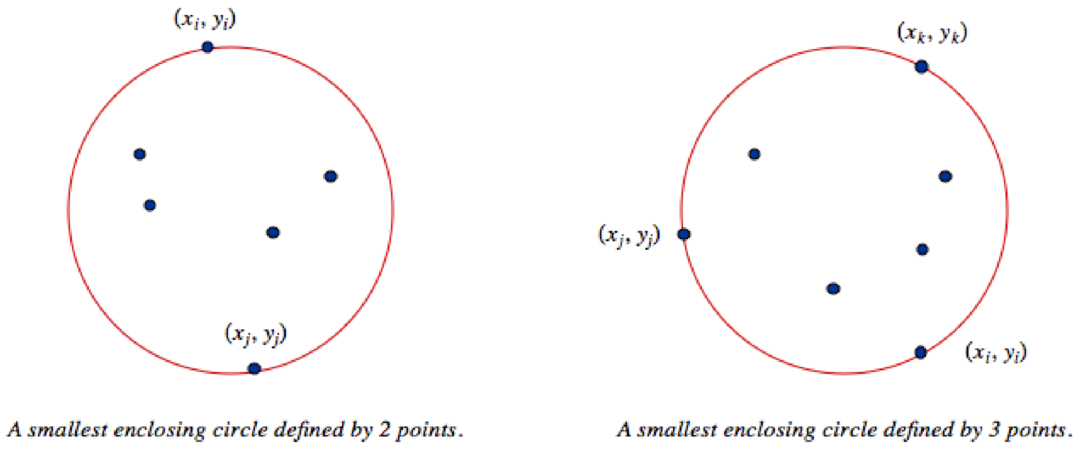

(x_i, y_i)
(x_j, y_j)
A smallest enclosing circle defined by 2 points.
(x_k, y_k)
(x_j, y_j)
(x_i, y_i)
A smallest enclosing circle defined by 3 points.

Figure 2.4: Smallest Enclosing Circle[4]

Now to find the smallest enclosing circle we first take the furthest two points in the set and draw a circle with these points as the diameter. The smallest circle will be either this circle or the circle with the minimum radius in C, whichever in minimum. The same is illustrated for both cases in Figure 2.4.
Alternate solutions maybe found by changing the definition of distance or changing the reference manifolds/facilities used. It is also possible to define an alternate problem which attempts to find the smallest enclosing convex space (or largest empty convex space in previous section). This can be looked at in detain in (4).

[4]Princeton CS

2.3 Optimal location of facilities in a distribution

So far we had assumed the facilities to be fixed and tried to find locations where we are at an optimum distance from the facilities. Now we look at the reverse scenario wherein we try to find the optimal location for facilities given population distribution in a region. This is precisely what we need in our problem to aid in optimum sensor coverage. We wish to find the location we should place the sensors in order to achieve the best coverage.

We will assume the number of partitions to be made are known. This is valid in our problem since we often know the number of agents we will using to aid in the task. We know will use the concept of weighted Voronoi partition to achieve the optimal configuration of the agents. In nearest point Voronoi configuration we only cared about the point facility to find the partitions. But now we need to find both these points (location for sensors) as well as the partitions (their coverage domains). Thus we need to consider the density function for the distribution in the space S. We will denote this as $f(p)$. The optimization is represented mathematically as,

$$\min_{s_i \in S} F(s_1, s_2, ..., s_n) = \min_{s_i \in S} \sum_{i=1}^{n} \int_{V_i} f(p) d_E(p, s_i) dp \tag{2.7}$$

In the above equation V_i denotes the Voronoi partitions as defined in Eq. 2.1. The solution to the above optimization will be discussed later in the controller designed section. Before we move on to other problems it is worth addressing the issue of computing the Voronoi partitions in a distributed manner onboard the bot. The next section is dedicated to this problem.

2.4 Computation of Voronoi partitions in a distributed manner

Consider n bots in a region $Q \in \mathbb{R}^N$. We assume that the bots will have wireless communication with at least the nearest neighbours (The neighbours which will determine the Voronoi partition for each bot). Further we assume that each bot knows its own location in space and can transmit it to any other bot (in its communication range) when requested for.

Suppose that each bot i located at p_i holds the information of the location of the other bots in a state vector P_i. Our task is to find the smallest radius R_i around p_i

such that we have enough information to find the Voronoi partitions. This agrees with the assumption that we need to have communication with only the nearest neighbours.

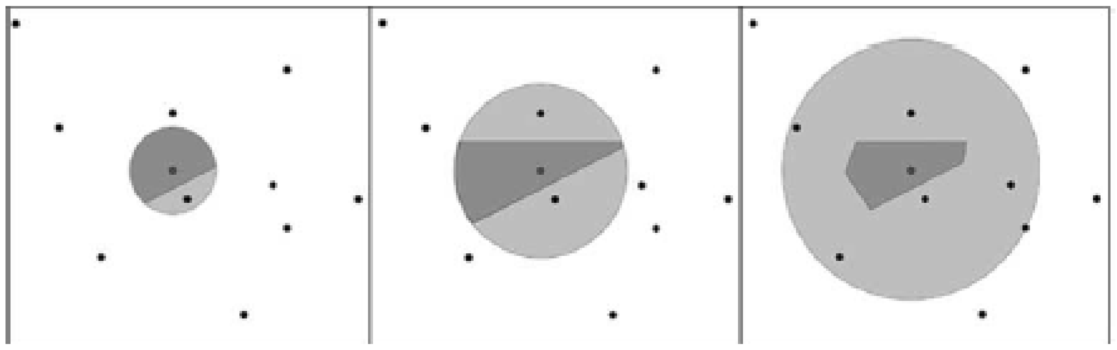


Figure 2.5: Adjusting Radius to find the Nearest Neighbours

Let $B(p_i, R_i)$ denote the n-dimensional ball of radius R_i centered at p_i. Let S_{ij} denote the halfspace defined by $S_{ij} = \{q \in Q\ S.T.\ ||q - p_i|| \leq ||q - p_j||\}$. Define the set $W(p_i, R_i)$ as follows,

$$W(p_i, R_i) = B(p_i, R_i) \bigcap (\bigcap_{j:||p_i - p_j|| < R_i} S_{ij}) \tag{2.8}$$

Know as we keep adjusting the radius we can add more points in defining halfspaces S_{ij}. If we stop increasing the radius when it is twice the maximum distance in between the point p_i and a vertex of $W(p_i, R_i)$ as defined in Eq. 2.8 we obtain the Voronoi partition. Mathematically when,

$$R_{i,min} = 2 \max_{q \in W(p_i, R_{i,min})} ||p_i - q|| \tag{2.9}$$

We get the Voronoi partition as $V_i = W(p_i, R_{i,min})$. The algorithm for obtaining the same is highlighted below from (5).

2.5 Simulation of the numerical algorithm

The algorithm proposed above was simulated on MATLAB® and the results of these computations are shown below. The circles indicate individual points in a grid. The sum of points is a simple measure of the area. The red crosses indicate the location of each bot. The different colors indicate the different Voronoi area partitions.

```
**Result:** Obtain the Voronoi partitions around $p_i$
initialize $R_i$
detect all $p_j$ within $R_i$
update $P^i(t_i)$, compute $W(p_i(t_i), R_i)$ **while**
 $R_i < 2max_{q \in W(p_i(t_i),R_i)} \|p_i(t_i) - q\|$ **do**
    set $R_i = 2R_i$
    detect all $p_j$ within $R_i$
    update $P^i(t_i)$, compute $W(p_i(t_i), R_i)$
**end**
set $R_i = 2max_{q \in W(p_i(t_i),R_i)} \|p_i(t_i) - q\|$
set $V_i = W(p_i(t_i), R_i)$
```

Algorithm 1: Algorithm for Voronoi Partitions computations

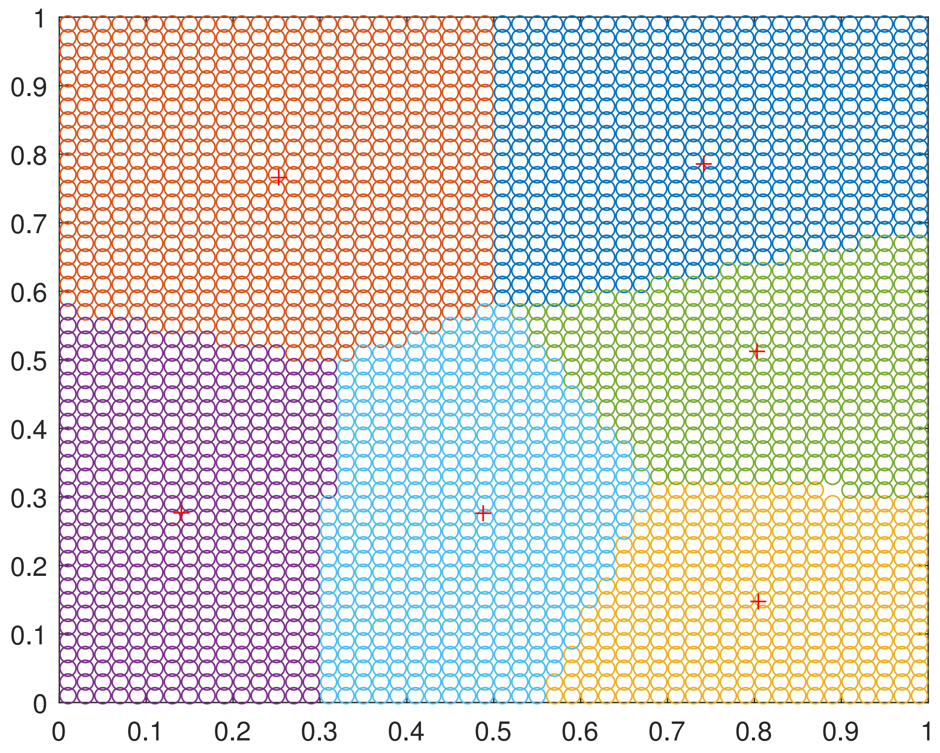


Figure 2.6: Area Voronoi partitions with 6 bots

Chapter 3

Fire Bird V Dynamics

The Fire Bird V bot by NEX Robotics is a non-holonomic robot with two fixed wheels and a castor wheel in the front. This may be illustrated as in the figure 3.2 below. The two fixed wheels are at a distance $2R = 0.15m$ and have a diameter of $2r = 5.1cm$. The wheels are controlled using a DC motor which is controlled using PWM inputs from a Atmega 2560 micro controller. We will attempt a dynamic level control of the bot and hence, we have to control the torque output of the motor.

$$\bar{M}(q)\dot{v} + \bar{V}_m(q,\dot{q})v + \bar{F}(v) + \bar{\tau}_d = \bar{\tau} \tag{3.1}$$

3.1 Torque control of the robot

PWM control works at a constant voltage and but varies the time interval over which that voltage is applied. As a result the average voltage results in a certain RPM. In steady state (No acceleration or rotational acceleration) the torque in the motor is balanced by the torque due to the friction force and rotational friction in the motor.

Transients will additionally have the acceleration terms as well as electrical transients due to inductance and resistance. We can thus obtain a correlation between the RPM, torque ,velocity and control PWM.The coupled electro-mechanical motor system is presented as a block diagram in 3.4. The electromagnetic torque T_m is proportional to the armature current and the back EMF V_b produced by the motor is proportional to the angular velocity of the motor. R and L indicate the resistance and inductance of the motor armature respectively. Finally J and b are the inertia and the damping present in the motor. The input to this system is the PWM in-

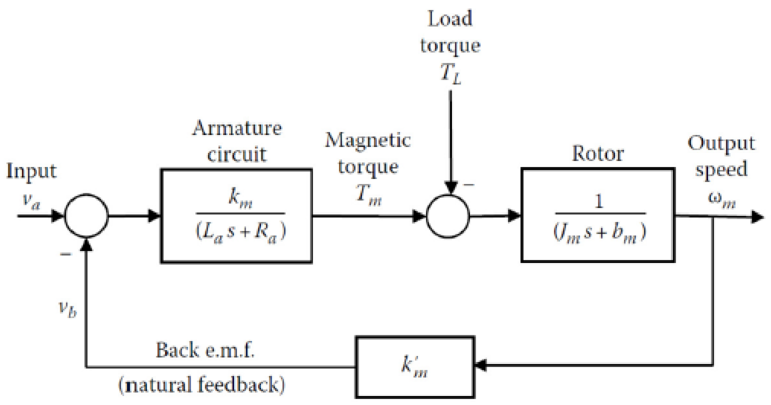


Figure 3.1: The block diagram for an electric motor

put voltage V_i and the output is the load torque which is directly our input to the non-holonomic bot equation. The equations governing the system are,

$$V_a = L\dot{i} + Ri + V_b \tag{3.2}$$

$$V_b = K_m\omega \tag{3.3}$$

$$T_m = J\dot{\omega} + b\omega + T_L \tag{3.4}$$

$$T_m = K_T i \tag{3.5}$$

We have the encoder values and hence can obtain the angular velocities of the bot. Further we can also obtain its higher derivatives. Since the bot had no current feedback it is difficult to create a very rigorous torque controller. We will combine the equations to by eliminating current to obtain,

$$\frac{J}{K_T}\ddot{w} + \frac{b+RJ}{K_T}\dot{\omega} + (\frac{bR}{K_T} + \frac{K_m}{L})\omega + \frac{1}{K_T}\dot{T}_L + \frac{R}{K_T}T_L = \frac{V_i}{L} \tag{3.6}$$

From our adaptive controller design for the optimal coverage algorithm we can obtain the desired torque T_{Ld} input as this is to be tracked. The error in this torque maybe denoted by $e = T_{Ld} - T_L$ and $\dot{e} = \dot{T}_{Ld} - \dot{T}_L$. We choose the V_i as follows,

$$V_i = L(\frac{J}{K_T}\ddot{w} + \frac{b+RJ}{K_T}\dot{\omega} + (\frac{bR}{K_T} + \frac{K_m}{L})\omega + \frac{1}{K_T}\dot{T}_{Ld} + \frac{R}{K_T}T_{Ld}) \tag{3.7}$$

Substituting this controller in the Eq. 3.6 we get exponentially stable dynamics as $Re + \dot{e} = 0$. This ensures that our desired torque control will be implemented.

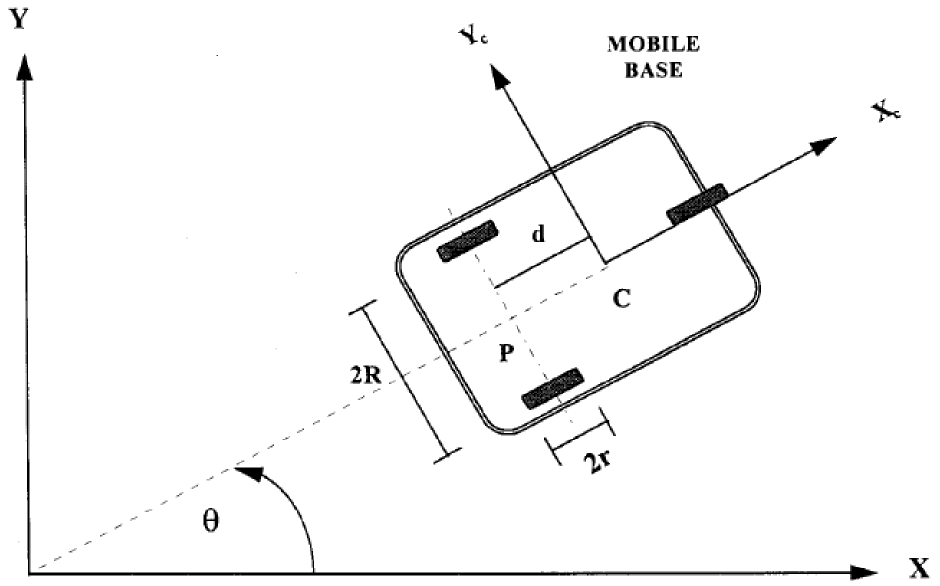


Figure 3.2: Non-Holonomic Robot

3.2 Computing motor angular velocity for feedback

We have set up the bot to send serial data of the bot encoder value every 0.04s or at 25 Hz using the Timer 4 built into the Atmega 2560 micro-controller. The encoder has 30 slots built into it and this would mean that each slot corresponds to an angular rotation of 12^o. With some basic experiments we arrived at the fact that the encoder actually undergoes 512 steps per revolution of the wheel resulting in a angular resolution of 360/512. If in a time interval of 0.04s we have x counts of the encoder signal we can approximate the angular velocity of the motors as,

$$\omega_d = \frac{x_d \times 360 \times \pi}{180 \times 512 \times 0.04} \; d \in \{L, R\} \tag{3.8}$$

The radius of each wheel is $r = 2.55\ cm$ and assuming both motors are rotating at equal angular velocities we can obtain the linear velocity of the bot as,

$$v = \frac{r \times x_d \times 360 \times \pi}{180 \times 512 \times 0.04} \; d \in \{L, R\} \tag{3.9}$$

This equation will be used purely as a validation of our system.

Chapter 4

Hardware Implementation

The Firebird V will be connected to the computer over a ROS network running on the onboard Raspberry Pi B+. The Raspberry Pi is running Raspbian Stretch (Raspbian Stretch Lite (CLI only) maybe used as well). The steps for the same are available on the **website**. The link also gives an explanation for installing ros-kinetic on the Raspbian platform.

4.1 Setting up SSH on Raspberry Pi B+

The steps here will be for a Linux (Ubuntu) computer communication over Ethernet cable or WiFi with a Raspberry Pi. We first enable the SSH option on the Raspberry Pi which comes disabled by default. Enter the following into a terminal (or directly into the CLI for Stretch Lite).

```
sudo raspi-config
```

Navigate to Interfacing Options and check the enable button next to SSH. Other basic setup such as connecting to WiFi and setting region maybe done now as well. Once this is done we need to set-up a static IP address on the Raspberry Pi so that we can always connect to it over a WLAN. The procedure below is done assuming that the Raspberry Pi as well as the master computer is connected over the same network through router. Note the usage of the word 'master' does not imply that the robots are receiving command inputs from the computer but only that this computer acts to process and send the position coordinates of the various bots to them over ROS. Into the CLI of Raspberry Pi we will first enter,

```
cat /etc/resolv.conf
```

```
# Generated by resolve.conf
# nameserver 192.168.43.1
```

The IP address attached to nameserver is the router address as seen by the RasPi. Next, we go ahead and setup the static IP by opening the file below with a text editor of choice,

```
nano /etc/dhcpcd.conf
```

Into the file we have opened enter the following lines,

```
interface wlan0
static ip_address=192.168.43.2/24
static routers=192.168.43.1
static domain_name_servers=192.168.43.1
```

Note that above the static routers and static domain_name_servers are set equal to the IP address obtained from cat /etc/resolv.conf. For the static ip_address we will set a value of our choice for the last number. Two possible values are given and the second is taken only if the first isn't available.
Now to connect the master computer to the RasPi over ssh we use a single line command, ssh pi@192.168.43.2. Answer to the subsequent question with a yes and enter the password you set the RasPi with in the initial configuration step. If you haven't changed the password yourself the default password is *raspberry*.

4.2 ROS communication between two computers

To establish communications between two different machines it is essential to ensure only one ROS_MASTER node exists. We will use a configuration in which the laptop connected to the VICON®motion capture system is the ROS master and each bot has a node which subscribes to the position data published by a publisher on the master computer.

To ensure that all computers on a network see only the master computer as the master we need to set the environment variable ROS_MASTER_URI in each computer (Each bot and the master). In our case this can be done with a command on a new terminal as,

```
export ROS_MASTER_URI=http://192.168.43.76:11311
```

To find the IP address of the master computer we can look up the IP under wlan0 on typing ifconfig into the terminal on the master computer. Note that by default the ROS_MASTER_URI is set to the localhost address as,

```
ROS_MASTER_URI = http://localhost:11311
```

Thus for our master computer this export step may be omitted. There is one additional step to be taken to ensure that the ROS network understands the IP each bot has been assigned. For the same we need to assign the ROS_PI environment variable using the static IP we assigned each bot in section 4.1. The example has been done assuming the IP we gave in previous section 4.1 as,

```
export ROS_IP=192.168.43.2
```

Having done all this we should be able to run[1],

```
roscore
rosrun rospy_tutorials talker.py
```

on our master computer. We then can receive the messages published by the Talker node on the RasPi by running the Listener node as,

```
rosrun rospy_tutorials listener.py
```

4.3 The ROS communication network

Add a figure from rqt-graph here which explains the network

There are four nodes operating on each RasPi apart from the master node that is operating on the host computer. This nodes are run by running a python script of the same name. Finally a roslaunch will be written to run all nodes simultaneously using one command. The nodes are listed below.

1. botInputOutput
2. torqueController
3. logger
4. viconData

[1]This assumes rospy_tutorials package has been downloaded from (6).

4.3.1 The *botInputOutput* node

This node is responsible for collecting the data from the encoder using Serial communication with the bot and publishing it over the topic encoderData. It also subscribes to the the motor PWM inputs over the topic pwmCmd and sends it to the bot over Serial interface. There is a C script projectCode.c on the bot microcontroller that takes and send packets containing the PWM inputs and encoder data respectively. These packets are designed in a certain and have to be sent in a format inspired by the XBee API packets. The bot takes Serial inputs in the format,

Byte 1	**Byte 2**	**Byte 3**	**Byte 4**
'A'	dir	pwmLeft	pwmRight

Table 4.1: PWM Command Data Structure

The first byte always has to be the character 'A' and it indicates the beginning of the byte. The second byte has to be a character dir that can take values in $['8', '5', '4', '6', '2']$. These indicate respectively forward, stop, left, right and backward motion of the bot. Sending a '8' sets both the motors to positive rotation and leads to the bot moving forward. Sending a '4' sets the left motor to negative rotation and right to positive rotation. Similarly we have rotations for right and backward motion. The final two bytes indicate the PWM Inputs to be sent to each motor. The bot checks if the character sent is 'A' and then subsequently accepts the next three bytes of data to interpret the signal.

The C code projectCode.c also takes the encoder data from the bot and sends it over Serial interface to the Raspberry Pi. For this the format to be followed is,

Byte 1	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**
'A'	encoderRH	encoderRL	encoderLH	encoderLL	checksum

Table 4.2: Encoder Data Structure

```
Result: Takes the encoder data from the microcontroller on the FireBird V
        bot and sends it motor PWM commands
initialize serial communication with bot
Function Main ():
    start PWM_command subscriber and callback at every data received
    start encoder data publisher
    while True do
        receive Serial encoder data as in Table 4.2
        re-obtain 16-bit encoder values
        check veracity using checksum
        compute time interval between two encoder data point
        publish encoder data and interval over publisher
    end
end
Function callback_function(PWM_command):
    send 'A'
    set motion direction of bot
    if abs(PWM_command) $\geq$ 255 then
        set PWM_command = 255
    end
    if 20 $\leq$ abs(PWM_command) $\leq$ lower_threshold then
        set PWM_command = lower threshold
    end
    send PWM_command to bot
    end
```

Algorithm 2: Communication node botInputOutput

The encoder data that the bot measures is a 16-bit unsigned integer and thus requires two bytes to be sent over the serial interface. Having received the 'A' character the ROS Node on RasPi then uses the next four bytes to obtain the current encoder reading on both motors. The high byte (H) and low byte (L) are summed up to get each encoder's reading. The final byte is a simple check-sum to check the veracity of the transmitted data. This is important as loss of a single byte in the encoder data can lead to large errors in the interpreted encoder data. Note that the PWM commands are sent to the bot at 10 Hz data from the on-board Raspberry Pi and the encoder data is received at 25 Hz. The interval between two incoming serial inputs is also computed. This interval is then used to numerically differentiate the encoder readings to get angular velocity, acceleration and jerk. This is done in the torqueController codes explained in chapter 6.

The botInputOutput also publishes the encoder data coming in over the topic encoderData and subscribes to the pwmCmd computed by the controller node. The botInputOutput.py code available at (7) launches the node above. Refer the comments in the code and the algorithm 2 for clarity on how the above algorithm is implemented.

4.3.2 The *torqueController* node

This node is essentially where the control law goes. It receives the encoder reading as feedback by subscribing to the encoderData topic. It subsequently computes the control PWM input required to achieve the desired system responses and publishes it over the pwmCmd topic. It implements the 3.1 defined in section 3.1. It also takes live position data from the viconNode and uses it to compute the Voronoi partitions. These node is the heart of the algorithm implementation and requires a whole section to itself in form of chapter 6.

4.3.3 The *viconNode*

The vicon_bridge package maybe downloaded or cloned from the GitHub from the ETHZ - ASL's repository. (https://github.com/ethz-asl/ros-drivers/tree/master/vicon_bridge). Doing so necessitates another step which involves setting the IP address of the PC publishing the VICON data in the vicon.launch file. The same is explained below.

Open the launch/vicon.launch file (Can be found in the vicon_bridge pkg.) in your preferred text editor. We need to edit the datastream_hostport variable to the IP address of the PC running the VICON Tracker software.

```
<param name="datastream_hostport" value="vicon:801" type="str" />
```

In our case (ARMS Lab) this becomes,

```
<param name="datastream_hostport" value="10.141.200.245:801" type="str" />
```

The above step maybe avoided by cloning the vicon_bridge package from this link instead (https://github.com/vijeth27/vicon_bridge). This package already has the launch/vicon.launch edited suitably. The next step is to launch the viconNode. We use the following command,

```
roslaunch vicon_bridge vicon.launch
```

As any other launch file it will begin a roscore node as well as publisher nodes which will publish the following topics.

- vicon/subject_name/segment_name
  publishes all available subjects/segments

- vicon/markers
  publishes all labeled and unlabeled markers. Labeled markers are not affected by origin calibration

Subscribing to these topics is covered in Chapter 6.

4.4 Cloning the the SD card for the Raspberry Pi

Re-installing the Raspberry Pi Software, setting-up SSH, installing ROS and copying all the files can be quite cumbersome. But we can reduce all of this into one single step by cloning the SD card image with the Raspbian installation into an empty SD card. There are numerous guides online to help us with the process. For the sake of completeness included here is the process to be followed for a cloning in a Linux environment.

4.4.1 Cloning an SD card in Linux

1. Insert the SD card in your PC using a USB or built-in card reader. Now open a Terminal window, and enter the command sudo fdisk –l. This will list all the filesystems present on your system.

2. Try to find out the device name of your SD card. I have a 16GB SD card, so it is easily identified as the device /dev/sdb which has a size of 14.9GB. This is because the actual storage on a device is always slightly lower than advertised. Note down this device name.

3. Use the dd command to write the image to your hard disk. For example:

   ```
   sudo dd if=/dev/sdb of=~/raspbian\_backup.img
   ```

Here, the *if* parameter (input file) specifies the file to clone. In my case, it is /dev/sdb, which is my SD cards device name. Replace it with the device name of yours. The **of** parameter (output file) specifies the file name to write to. I chose raspbian_backup.img in my home directory.

You can now remove the SD card and use it in your Pi. Once you are ready to restore the backed up image, follow the instructions below:

1. Insert the SD card in your PC. Before we restore the image, it is important to make sure that the SD cards partitions are unmounted. To verify this, open the Terminal, and execute the command sudo mount | grep sdb. Here, replace sdb with your SD cards device name.If you see a blank output, you do not need to do anything. If you do see some mounted partitions, unmount the listed ones using sudo umount /dev/sdb1.

2. Use the dd command to write the image file to the SD card as,

   ```
   sudo dd if=~/raspbian_backup.img of=/dev/sdb
   ```

   This is like the command we used to make a clone, but reversed. This time, the input file if is the backup image, while the output file of is the SD card device. Verify, and double-verify the parameters here, as entering the wrong command here will cause permanent data loss.

3. Once the write is complete, you will see a confirmation from dd. You can then remove the card from your PC, and insert it back in the Raspberry Pi.

Note: Different SD cards have different amounts of free space when empty depending on the maker of the SD card. Ensure to use the same make and same size of the SD card when burning SD card image file. SanDisk 16 GB was used for the project.

Chapter 5

Motor System Identification

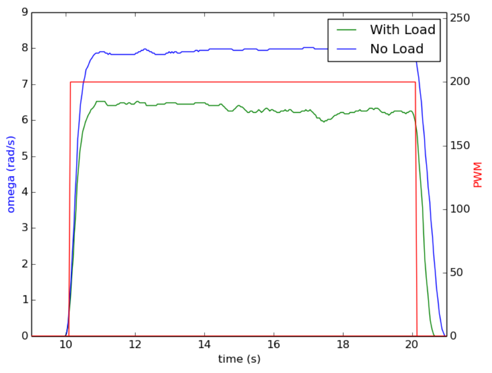
To obtain torque level control of a FireBird V using DC motors we proposed a system model and a subsequent control law for the trajectory tracking in section 3.1. In this section, we will obtain the the parameters of the system mentioned above. (4)

5.1 Verification of the encoder precision of the motor

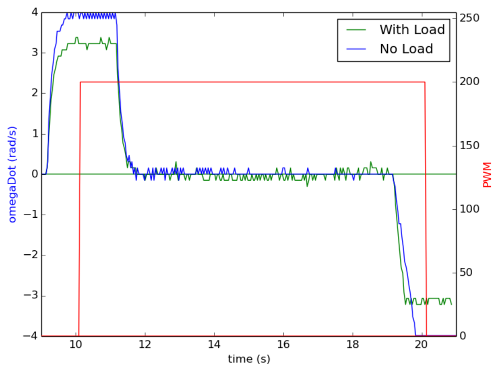
The manual for the FireBird V gave the encoder precision at 30 counts per revolution. But this led to inaccurate angular velocities as observed in physical experiments. To resolve this an experiment was conducted where a fixed PWM input (120) was fed into the right motor and the number of rotations were observed. In a $10s$ interval 8.78 $(8 \times 2\pi + 280^o)$ rotations of the wheel took place. The encoder value at the end of these rotations were 4495. A simple division results in 512 counts per revolution of the wheel.

5.2 Step, Ramp and Sine Input Responses

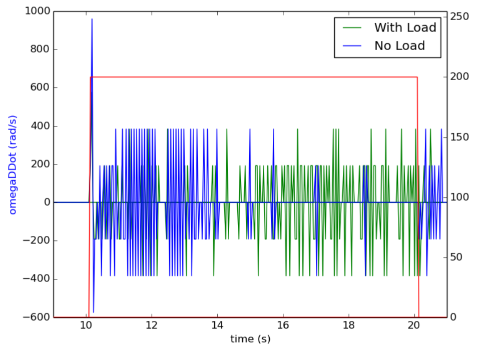
The step responses for the right motor were obtained by giving a constant PWM input of 200 to the system for 10s. This was conducted both when the bot was off the ground as well as when it was kept on the ground. The friction forces of the ground of-course will cause a slower rotation for the same PWM input when the bot is kept on the floor. The step responses corresponding to both the scenarios have been plotted below. Note that due to the numerical nature of differentiation the noisy encoders result in large fluctuating values of the angular acceleration. This accelerations have been smoothed out by taking an average of 10 samples around each instant. This results in the angular acceleration appearing to rise a fraction of a second before the input. The choice of number of smoothing samples was made based on the the desirable ambient noise levels in the measurement as well as considering the loss in accuracy. We intend on using regression based curve-fitting on the original data before smoothing as WLS regression is in essence an optimum smoothing function. The smooth for the graphs is done only for easy in viewing and interpreting.



(a) Step response of angular velocity



(b) Step response of angular acceleration

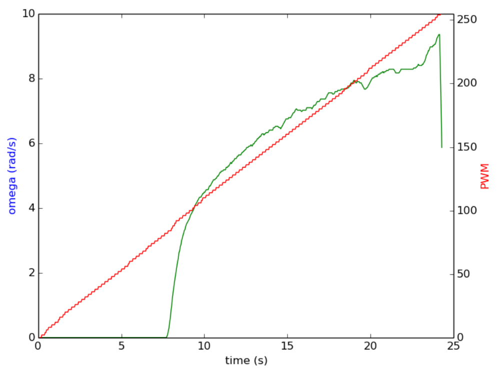


(c) Step response of angular jerk

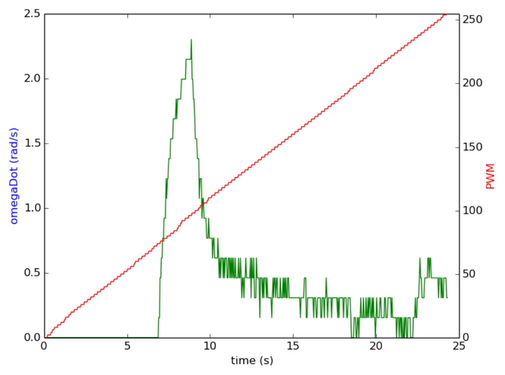
Figure 5.1: Step Response

In Fig. 5.1c we see that the magnitude of noise in the system is larger than the value of the measurement itself. Thus, the jerk results are neglected in fitting the data. Note that for jerk response we have plotted the results without smoothing to illustrate the actual magnitude of disturbances.

Shown in 5.2 is the response of angular velocity and acceleration of the motor



(a) Ramp response of angular velocity under no load



(b) Ramp response of angular acceleration under no load

Figure 5.2: Ramp Response

to a ramp input. This graph highlights a feature of a digital circuit powering a DC motor using a PWM input. Consider the motor controller H-Bridge in Fig. 5.3. When Q1 and Q2 are closed leaving Q2 and Q3 open we have rotation in one direction and the opposite rotation in the vice-verse scenario. When we have rapid switching in a PWM input, where the switches Q1 and Q4 close for a fraction of a sampling time and open again, there is a large back EMF generation. This back EMF is greater than the magnitude of average torque and thus there is no rotation in the motor. This leads to a zero magnitude of angular velocity until a threshold (close to 80) is reached. This effect drops when we cross the threshold of 80 (in PWM) and we will assume our region of operation to be above 80 (in PWM) for the major portion of the time.

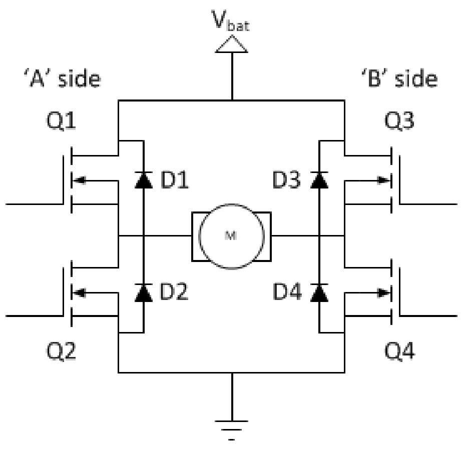
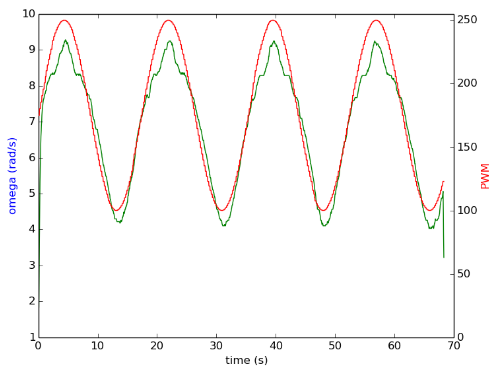
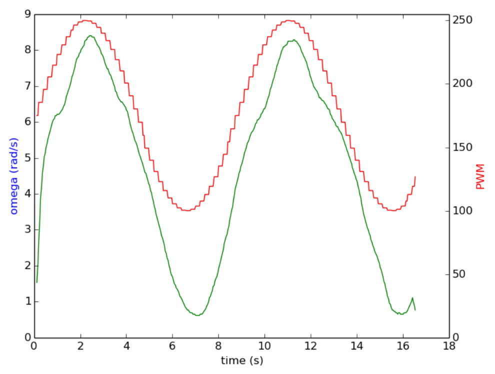


Figure 5.3: Control of a DC motor using H-Bridges

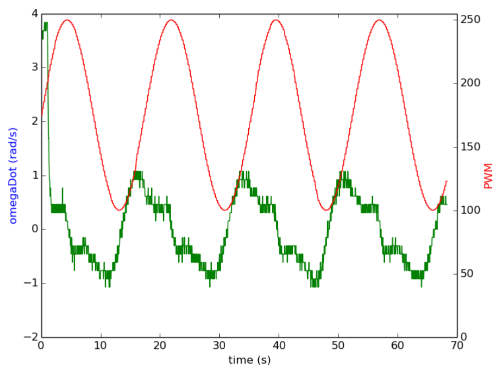
Finally, we will look at the effect of sinusoidal PWM inputs to the motor in presence and absence of any load. This data will be used for the fitting the system identification parameters using regression.



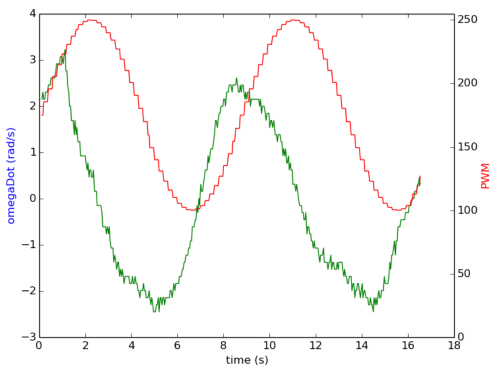
(a) Angular velocity response in absence of Load



(b) Angular velocity response in presence of Load



(c) Angular acceleration response in absence of Load



(d) Angular acceleration response in presence of Load

Figure 5.4: Sine Wave Response

5.3 Parameter Identification

$$V_i = L(\frac{J}{K_T}\ddot{w} + \frac{b+RJ}{K_T}\dot{\omega} + (\frac{bR}{K_T} + \frac{K_m}{L})\omega + \frac{1}{K_T}\dot{T}_L + \frac{R}{K_T}T_L) \qquad (5.1)$$

Due to the high noise levels in the jerk data for angular rotation we will neglect the contribution of the first term in Eq. 5.1 above. Further, since we can measure the resistance of a motor easily using a multimeter, we can obtain the coefficient for the second last term if we know knowing the coefficient for the last term. Since only steady state T_L is required we can compute this quantity from the differences in step response in cases with and without load on the wheels. Further the quantity T_L is simply due to rolling friction between wheels and the ground. The rolling friction coefficient in a wheel is a function of its slip ratio. With slip ratio being defined as,

$$S = \frac{\omega R}{V} - 1 \tag{5.2}$$

At low values of slip ratio (< 0.1) the coefficient of friction is a linear function of S. We will assume the torques of the DC motors don't lead to drastic slipping and assume $S < 0.1$. Thus, our T_L is a linear function of the slip ratio S as,

$$T_L(V, \omega) = \mu N = \mu_S(\frac{\omega R}{V} - 1)mg \tag{5.3}$$

Typical values of μ_S for rubber on asphalt are 2-2.5 and we will assume the same value. With the the above equation and from measurements we can easily estimate T_L. In the no load condition we have the following equation

$$V_i = L(\frac{J}{K_T}\ddot{w} + \frac{b + RJ}{K_T}\dot{\omega} + (\frac{bR}{K_T} + \frac{K_m}{L})\omega \approx \frac{b + RJ}{K_T}\dot{\omega} + (\frac{bR}{K_T} + \frac{K_m}{L})\omega \tag{5.4}$$

A regression model fit over the 5.4 gives us the following values,

$\frac{b+RJ}{K_T}$	$(\frac{bR}{K_T} + \frac{K_m}{L})$
0.088508	0.0002705

Table 5.1: Regression based values

Work ahead will involve estimating the slip ratio of the wheel to get estimates of the parameters associated with load torque.

Chapter 6

Codes Explained

This sections tries to give an an algorithmic overview of the project codes. It is divided into three main sections, explaining essential three codes and their purpose. All these codes are available at (7).

- The first is a test code to check the interface between the AVR code on the bot, the botInputOutputCode.py explained in section 4.3.1 and the commands being sent from a controller code.
- The second is an implementation of the Eq. 3.7 in a circular trajectory tracking problem.
- The final code, as its name suggests, is the final project code which involves the algorithm to achieve the optimal coverage configuration. **As of yet it does not include the code to adaptively arrive at the correct phenomenon distribution. It works from a known distribution case and moves to the corresponding optimal coverage locations.**

6.1 torqueControllerTest.py

This code maybe used as a simple test of communication through the ROS network of the bot and the RasPi. It has to be used in parallel with,

- botInputOutput.py (refer section 4.3.1)
- FireBird V running the projectCode.c (refer (8))

This code accomplishes three primary things - Introducing how data in stored on board the bot, sending PWM commands over a publisher and subscribing to encoderData being published on its topic. Algorithmic explanation of the code in brief is given at the end of this section. It maybe referred before viewing the code for a faster understanding of it's functioning.

6.1.1 Subscribing to encoder data

The topic encoderData published by the the botInputOutput node has the current encoder counts and the interval between the current and the previously obtained encoder data. A simple numerical differentiation here gives us the angular velocities, accelerations and jerks of the wheel. From the plots shown in the section on parameter identification, we observed that jerks are extremely unreliable and are zero for the most part of the motion. We will continue to ignore the values of jerk as earlier. These velocities and accelerations will be stored in global variables to aid us in logging and usage in other parts of the codes.

6.1.2 Further insight on logging methods

The most common methodology of logging, writing to a .csv file, has been used throughout the project. While efficient ROS logging methods are available, the used method was found to be effective and simple. It creates a single log file which can then be processed quite easily using python scripts. In addition to the created log file we will use another two scripts,

- dataPull.sh (refer (7))
- DataPlotting.py (refer (7))

For the purpose of this code, the data to be logged is restricted to timestamp, encoder data and PWM commands at each instant. As such a column-wise header line is created of the variables to be noted and is inserted at the top of ever log file to be created. The variables to be noted are given in the table 6.1.

Logging is done at a reasonable rate by placing the logging step logically at some point the script. In this script is is placed in the callback function for the encoder data. Since this callback occurs at a frequency of roughly 25 Hz by design of AVR C code, the logging is done at the same frequency here. The logger stores the data it logs in a CSV file which then can be used to obtain plots. This logger is what helps us in the system identification part of the project where we try to obtain the motor parameters. The node by default saves a data.csv file in the folder from where the

Columns	Field	Description
1	time	System time when the data was received.
2	interval	Time interval between reception of two data points.
3	encR	Right encoder count data.
4	encL	Left encoder count data.
5	wR	Angular velocity of right motor.
6	wL	Angular velocity of left motor.
7	wdotR	Angular acceleration of right motor.
8	wdotL	Angular acceleration of left motor.
9	wddotR	Angular jerk of right motor.
10	wddotL	Angular jerk of left motor.
11	pwmR	PWM input to right motor.
12	pwmL	PWM input to left motor.

Table 6.1: Logged variables

ROS nodes are run. This may be exported to the master computer using a simple scp command. This is further explained in section 6.1.3.

6.1.3 Extracting the logging information.

This section seeks to provide some insights on the two scripts below,

- dataPull.sh (refer (7))
- DataPlotting.py (refer (7))

The first shell script is a two line script which logs into the RasPi over SSH, copies the data.csv file from it's location to a particular location on the host computer and runs the second python script. The locations can be changed in the file. The second script takes the data.csv log file and extracts the data into Panda dataframe in python. This can then be used to generate graphs and plots of all kinds to better visualize the bot experiment data. Again, the code is quite simple to follow and can be understood by anyone with a basic idea in data-frames. One point of interest is the python script is the usage of smoothing functions for better visualisation of the encoder data. The reason for this maybe found in section 5.2.

```
Result: Sends PWM commands, subscribes to encoder data, logs data

import relevant packages and messages
define the header line for logging file
write the header line in a newly created logging file

Function callback_function(encoderData):
    compute angular position of wheel from encoderData
    compute angular velocity                  // Numerical diffrentiation
    compute angular acceleration
    compute angular jerk

    write required data into log file
end

Function Main ():
    start encoderData subscriber and callback at every data received
    start PWM Command publisher

    declare time step of input (timeStep)
    while True do
        i=0
        while i≤ 5 do
            publish PWM Command (50 + 50*i, 50 + 50*i)
                                                  // (R Motor, L Motor)
            hold PWM Command over the next timeStep time
            i=i+1
        end
    end
end
```

Algorithm 3: Algorithm for torqueControllerTest.py

6.2 torqueControllerTracking.py

In the previous code we tested the ability of the RasPi to send torque commands to the bot and to process the encoder data coming back from the bot. We also saw the logging framework for the bot. We will now proceed to add two more features - incorporating data from VICON system and the dynamical model for converting torques to PWM input through Eq. 3.7. The purpose of the codes is to check this two features exclusively and does no coverage or sensing functions. The *torqueControllerTracking.py* code requires in parallel with it the following three peripherals to be run.

- botInputOutput.py (refer section 4.3.1)
- FireBird V running the projectCode.c (refer (8))
- vicon.launch file from vicon_bridge pkg (refer (9))

More information about the last file and how to initialize and launch it can be found in section 4.3.3. We will first address obtaining the VICON data and using it appropriately.

6.2.1 Subscribing to VICON data

VICON data is published in the form of the geometry_msgs/Transform.msg messages (10). The structure of each Transform.msg is as given below,

```
# Represents the transform between two coordinate frames in free space.

Vector3 translation
Quaternion rotation
```

Translation gives us the position and rotation gives us the orientation of any object kept in the field of view of the VICON setup. For our purpose we need the heading angle (Φ) of the bot as well as the (x,y) position. To obtain the heading angle we will do an Quaternion to Euler angle conversion. For this a function has been written quat2eul(). These variables x,y and phi are again stored in global variables for logging and accessibility purposes. One major challenge ahead is to get accurate estimates of the velocity of the bot using VICON data. This has to be worked on.

The algorithm for subscription is given below,

```
Result: Subscribes to VICON data and returns the states x,y and $\Phi$

Function callback_function(VICONdata):
    x=data.transform.translation.x
    y=data.transform.translation.y
    eulerAng=quaterion2euler([data.transform.rotation.x,
      data.transform.rotation.y, data.transform.rotation.z,
      data.transform.rotation.w])
    /* There is a self written Quaternion to Euler angle
       converter in the code                                  */
    $\Phi$=eulerAng[0]
end
```

Algorithm 4: Algorithm for torqueControllerTTracking.py

We have implemented a non-linear back-stepping code to achieve tracking in this code. The controller was obtained from (11). We need to define the matrices involved in obtaining the dynamics of the bot. These are listed below,

- $S(q)$
- $\dot{S}(q)$
- $M(q)$, $\bar{M}(q)$
- $V(q, \dot{q})$, $\bar{M}(q, \dot{q})$
- $\bar{B}(q)$, $B(q)$

We will use the above defined matrices to define our controller torque to motors τ. This is then converted to PWM commands using Eq. 3.1. We use the constants obtained at the end of Chapter 5 through linear regression methods to design this control. The trajectory to be tracked is a circle in this code. A function is used to compute the trajectory at each point. Sections 6.1.2 and 6.1.3 can be referred to understand the logging in this code as well but with changes to the variables being logged.

6.3 torqueVoronoi.py

This is the final working code of the optimal coverage control of using 4 bots. This code causes the bot to the converge to their optimal positions assuming the entire distribution of the phenomenon to be sensed is known. The adaptive update law required to estimate the distribution of the phenomenon requires the velocity of the bot and a more robust ,method to obtain that has to be attempted. Chapter 8 explores some potential ideas that may work and may be tried out in the future.

This code adds a methodology to obtain the the Voronoi partitions for the configuration of the bots. While we can find packages that readily compute the Voronoi partitions given a certain set of points, what we need to compute is area integrals over the partitions. To do this we divide the area of the experiment into a fine grid and find the corner points of each square grid that lies within a specific partition. The same has been illustrated as an algorithm in Algo. **??**. Computation of the area integrals may be done after we compute the points in each partition using the method above. We then use the controller listed in (1) in place of the controller algorithm used in the preceding section.

The most important of this code is it's discretized nature. To dwell further on it's explanation it is best to work with the comments in the code as well as Wiki feature on GitHub. For the best overview of this code it is best to refer to (7). The time required in computation of Voronoi partitions, as well as the requisite area integrals forces, us to halt momentarily while processing this information. This, stop-and-go method has been tried and tested to work in the case where we assume we know the distribution beforehand. This also introduces additional challenges in computing the velocities of the bot. All of this has been dealt very elaborately with in the Wiki section on GitHub.

Sections 6.1.2 and 6.1.3 can be referred to understand the logging in this code as well but with changes to the variables being logged. But the scripts being used in this case are,

- dataPullVoronoi.sh (refer (7))
- DataPlottingVoronoi.py (refer (7))

Another clever hack is to create a shell script to log into all the bots involved in the simulation simultaneously. This can save both the time and effort of logging into each bot manually and navigating to the package containing the scripts. This has been achieved with the script sshBots.sh in (7). Change the IP addresses of each bot in the script to achieve the objective.

Chapter 7

Results of the torqueControllerXX.py codes

Here we have presented the results of the codes described in Chapter 6 only. The results of other chapters are self contained. There are more elaborate discussions on the GitHub link for these codes listed at (7).

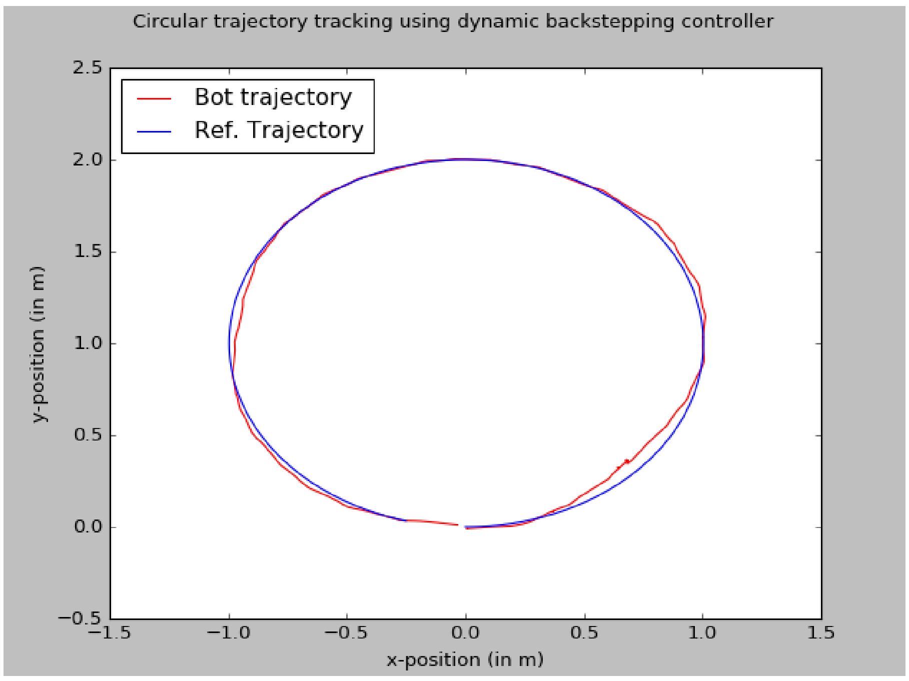


Figure 7.1: The circular trajectory is closely tracked by the bot using a dynamic backstepping controller. torqueControllerTracking.py is responsible for this performance when run on bot 0. Different bots have marginally different motor and wheel parameters and thus all bots will not yield the same performance with this code.

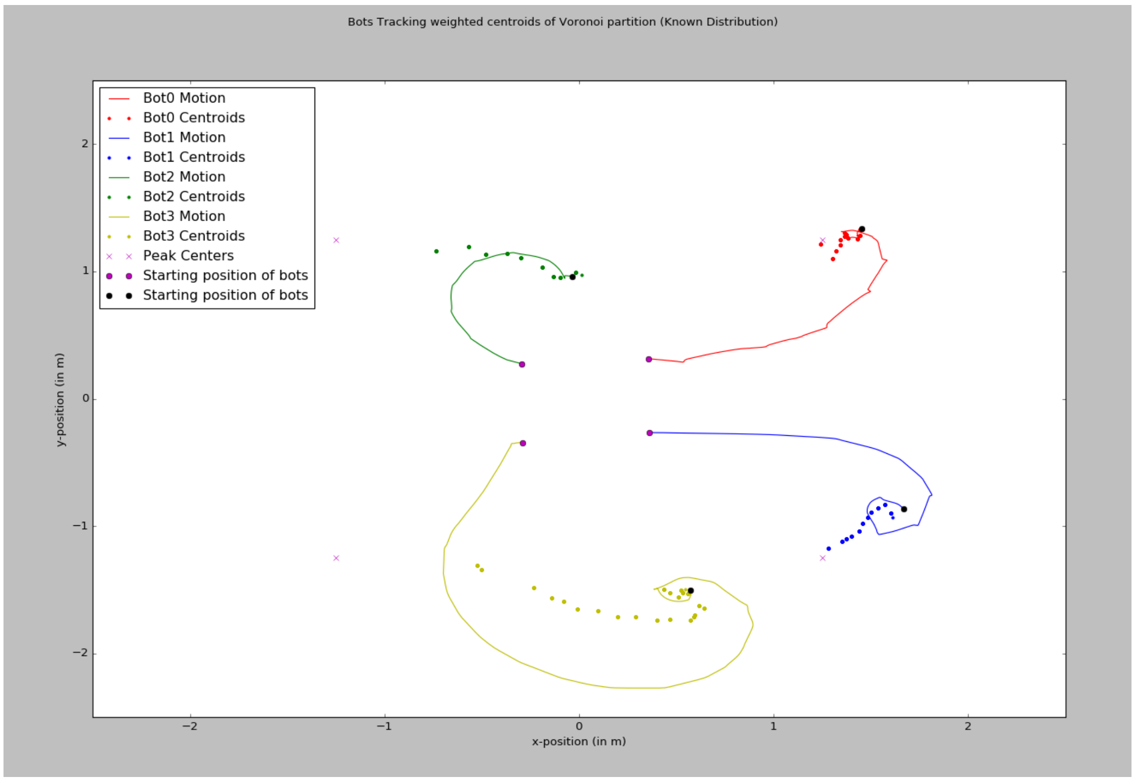


Figure 7.2: The four bots from the four locations marked by the magenta dots. They iteratively compute the centroids they have to move towards at discrete instances of time. These are denoted by the RGBY dots for the bots following the RGBY trajectories respectively. The magenta crosses denote the means of the four Gaussian basis functions. In this case two peaks (bottom-left and top-left) have been assigned zero weights, while the other two peaks have been assigned equal positive weight. The black dot indicate the final position of the bots in the optimal coverage configuration.

One can also obtain the experiment videos of the bots converging to the final configurations described above, on YouTube. The links for the same may be founded on the GitHub website at (7).

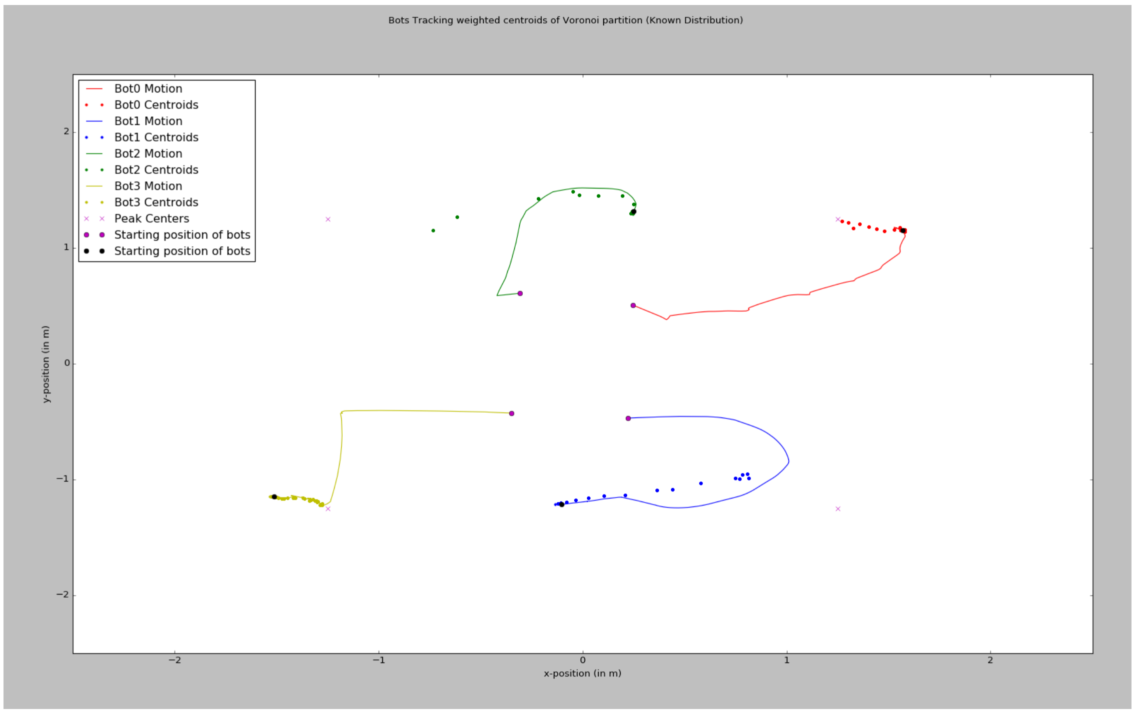


Figure 7.3: The four bots from the four locations marked by the magenta dots. They iteratively compute the centroids they have to move towards at discrete instances of time. These are denoted by the RGBY dots for the bots following the RGBY trajectories respectively. The magenta crosses denote the means of the four Gaussian basis functions. In this case two peaks (bottom-right and top-left) have been assigned zero weights, while the other two peaks have been assigned equal positive weight. The black sot indicate the final position of the bots in the optimal coverage configuration.

Chapter 8

Further direction

The major step ahead would be to obtain the accurate velocity of the bot for feedback. To do the same the following methods have been tried with poor results,

- We use the encoder data to obtain the wheel velocities, and subsequently use the matrix $\bar{B}$ to transform the wheel velocities into bot linear and angular velocity. This was discarded due to large noise in the readings from the sensors.
- Another way to compute velocities was to differentiate the position and attitude data obtained from the VICON setup. But, this fails as well. The reason for this could not be pinpointed, but this resulted in 0 magnitude for the velocities all the time.

A good method that may work for our case is usage of the average velocity in each instance of motion. We compute the position before we start motion and final after we stop motion. During the stopped state we will compute the Voronoi partitions as explained earlier. The time required for motion after computation is a fixed value and is known. We can obtain the average velocity in this instance of motion by taking the ratio of total distance moved by the total time for motion. This may lead to good readings for the velocity.

It is also a good idea to separate out the code that computes the PWM commands from the torque requirements from the torque computing code. This maybe incorporated into the code listed in 4.3.1. Refer to the GitHub link at (7) to more update ideas on how to proceed further and obtain better results.

Chapter 9

FAQs and Recurring Problems

- **SD card image burn is returning 'Not enough free space'**
  Different SD cards have different amounts of free space when empty depending on the maker of the SD card. Ensure to use the same make and same size of the SD card when burning SD card image file. SanDisk 16 GB was used for the project.

- **Quickly accessing SSH terminals in all the bots.**
  Another clever hack is to create a shell script to log into all the bots involved in the simulation simultaneously. This can save both the time and effort of logging into each bot manually and navigating to the package containing the scripts. This has been achieved with the script sshBots.sh in (7). Change the IP addresses of each bot in the script to achieve the objective.

- **Problems with running torqueControllerVoronoi.py codes due to multiple master node**
  Remember that the vicon.launch file in vicon_bridge package automatically initializes a ROS Master node. Running another Master node in parallel can cause problems and confusions in the ROS network. Close the other Master node and run the codes again.

- **Orientations of the bot are wrong when placed in the VICON field**
  When creating the object make sure the initial orientation of each bot is correct. In the Vicon lab (10x8 mat layout) the center is at the 5x4 location. Further the x-axis is along the direction towards the main door of the lab and the z-axis is upwards. y-can be obtained for the right handed system. When creating a VICON object ensure the object x-axis is oriented along the

VICON x-axis. Else, additional calibration steps have to performed.

Refer to the GitHub link at (7) for a more elaborate debugging and FAQ list.

Bibliography

[1] Rihab Abdul Razak, Sukumar Srikant, Hoam Chung *"Decentralized and adaptive control of multiple nonholonomic robots for sensing coverage"*. Int. J of Robust Nonlinear Control, 2018

[2] Mac Schwager, Daniela Rus, and Jean-Jacques Slotine *"Decentralized, Adaptive Coverage Control for Networked Robots"*. 2008

[3] R. Fierro and F. L. Lewis, *"Control of a Nonholonomic Mobile Robot: Backstepping Kinematics into Dynamics"*. Journal of Robotic Systems, 1997

[4] Atsuyuki Okabe, Atsuo Suzuki *"Locational optimization problems solved through Voronoi diagrams"*. European Journal of Operational Research, 1998

[5] Jorge Corts, Sonia Martnez, Timur Karatas and Francesco Bullo *"Coverage Control for Mobile Sensing Networks"*. IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, 2004

[6] ROS Tutorials, *http://wiki.ros.org/ROS/Tutorials*

[7] Python codes on RasPi, *https://github.com/vijeth27/fb5_torque_ctrl*

[8] C codes on AVR, *https://github.com/vijeth27/FirebirdV_BTP*

[9] VICON Bridge modified for ARMS Lab, *https://github.com/vijeth27/vicon_bridge*

[10] Transform message type, *(http://docs.ros.org/melodic/api/geometry_msgs/html/msg/Transform.html)*

[11] Yasmine Koubaa, Mohamed Boukattaya, Tarak Dammak, *"Adaptive dynamic tracking control of uncertain wheeled mobile robot including actuator dynamics"*, Sciences and Techniques of Automatic control computer engineering, 2016