# IMPROVING TIME MODELING OF TURTLEBOT

Rajat Deepak Mathur
Institute for Software Research
Carnegie Mellon University

## ABSTRACT

The project aims at better understanding motion for Turtlebots and make time estimates for an instruction fed to the turtlebot . A robotic system's architecture consists of the entire stack of hardware used and software executing on it. This project focused primarily on the software aspect of a robotic system and how that impacts the motion. Experiments were conducted on a simulator for a Turtlebot and observations of time, speed and position were made. This was used to break down robot linear motion into multiple stages and creating an empirical model. The model proposed calculates the total time taken by turtlebot to complete a linear motion given the desired speed in that direction and the distance between the source and the destination.

## INTRODUCTION

### TURTLEBOT

The Turtlebot is a low cost personal robot tool kit and consists of a 3D sensor, a mobile base, mounting hardware and a laptop. Turtlebot can drive around an indoor environment and can map the environment using the 3D sensors mounted on it. Turtlebot 2 consists of Kobuki base, a 2200mAH battery, a Kinect sensor and a module plate to mount to computer. The software stack consists of an SDK for Turtlebot, desktop environment and libraries for visualization, planning , perception, control and error handling and also some demo applications.

### ROS DISTRIBUTED ENVIRONMENT

ROS which stands for Robot Operating System is a middleware technology compatible with most Linux flavors. It aids in distributed computing and is very popular in the robotics domain. ROS provides many services for hardware abstraction, low-level device control, message passing and package management. ROS's development was done keeping in mind the distributed and iterative development of software in robotics. It helps roboticists to push new packages with varying functionality without caring much about continuity of service or downtime for the system. This helps multiple teams working on planning, mapping, vision, motion and perception to work collaboratively on the same hardware system. ROS has many open source implementation of common robotics functionalities and algorithms. These are called packages and most of these packages come coupled with the standard ROS distribution.

### BRASS TEAM'S MARS ENVIRONMENT

CMU's BRASS team has developed an headless Ubuntu environment for testing of ROS nodes. This can be used as a testbed before deploying on the actual physical robots. The MARS environment bundled as a Ubuntu VM consists of ROS Indigo middleware along with it's default packages for navigation and movement, instruction graph server package, gazebo robot simulator, mars notification package, maps of some CMU blocks, instruction set and launch files to start up multiple nodes.

## INSTRUCTION GRAPH SERVER

Ig_server is a ROS package developed at Carnegie Mellon University and is executed in the MARS environment. It is used to manage and launch the ROS Gazebo simulator. The ig_server reads the set of instructions needed to navigate through a map and executes specific move_base commands on parsing each instruction. The instruction graph server is a ROS node named ig_action_server and publishes status message (current goal, goal completion, error messages). The launch file subscribes to these messages and prints them to the console.

## NAVIGATION PACKAGES

The TurtleBot uses the move_base package from the navigation packages available for ROS environment. This package gives an implementation for the goal fed to it by the actionlib package. move_base node links together a local and global planner to accomplish the goal and attempts to reach the goal with the mobile base of the robot. The move_base and the mobile base take considerable amount of time in reaching the desired full speed of the system.

## PRISM BASED TIMING MODEL

The PRISM model predicts the estimated amount of time for an instruction graph. It can handle translational and even rotational motion. The entire instruction graph is broken down into set of instruction between many nodes and timing estimates for each section are made. The model doesn't account for the duration of motion when the turtlebot is moving at a speed lower than the desired full speed.

## ODOMETRY

Odometry is a technique to model the motion of robots in a 2D plane. A robots position can be described using 6 parameters: three-dimensional Cartesian coordinates plus three Euler angles pitch, roll, and tilt. In our case in a 2-D plane, we can work with just 3 coordinates: <x, y, θ>: where x, y represent the position and θ: the orientation.

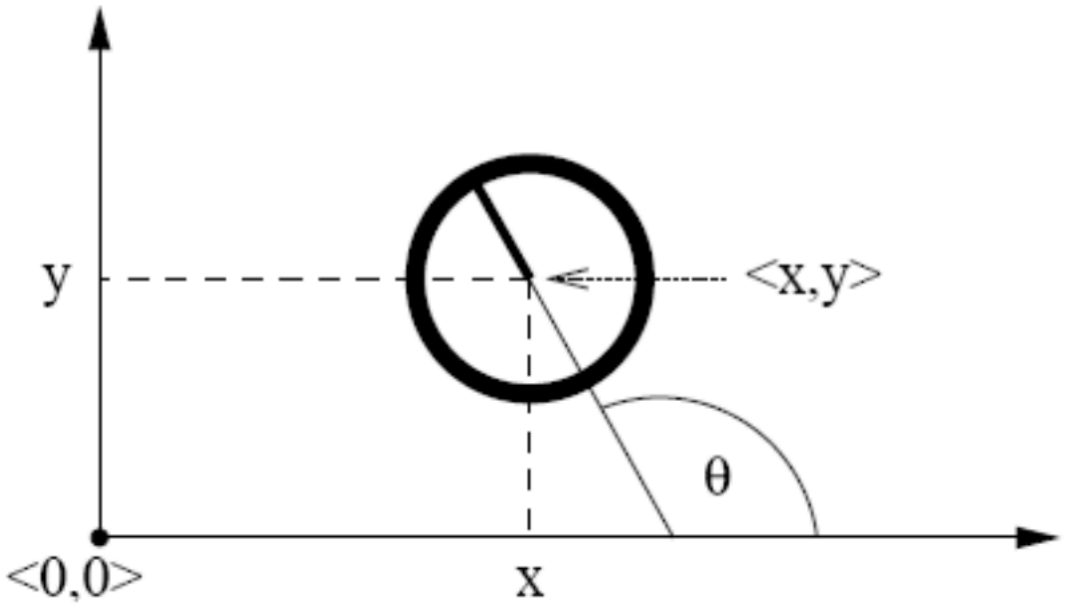


**FIGURE 1: 2D CARTESIAN PLANE**

Robot motion modeling can be velocity-based or Odometry-based. In the former, the wheel has no encoders to collect instantaneous speed and position. The new position is calculated using velocity and the time elapsed. Odometry-based modelling requires encoders in the wheels of the robot to tell about position and instantaneous velocity.

A linear encoder is a sensor, transducer or readhead paired with a scale that encodes position. The sensor reads the scale in order to convert the encoded position into an analog or digital signal, which can then be decoded into position by a digital readout (DRO) or motion controller [2].

Robot moves from to <x, y,θ> to <x', y',θ '>. For the experiments conducted for this study:

$y = y'$

$\theta = \theta'$

# PROBLEM AND GOAL DESCRIPTION

## LIMITATIONS OF THE CURRENT TIMING MODEL

There is a lack of handling non-constant speed motion. The entire instruction is assumed to be executed at constant speed matching the desired speed fed as part of the instruction. This however is not true in practice, since the Turtlebot takes a variable amount of time for ramp up motion. It also has gradual increase and decrease in speed during the ramp up and ramp down sections. A portion of the total distance is thus covered at less than the desired speed.

# PROPOSED SOLUTION

The proposed solution tries to break down the entire Turtlebot motion into 3 stages: Ramp Up Motion, Constant Speed Motion and Ramp Down Motion. Each instruction will be provided with the desired speed and the total distance to be covered.

## RAMP UP STAGE

This is the duration of time from the instant the simulator receives the instruction from the ig_action_server to the time instant the Turtlebot reaches desired speed for the first time. The Odometry messages are logged for this purpose and manually searched to find the first occurrence of the desired speed. No significant speed overshoots and undershoots were observed for the motion once the desired speed has been reached. The motion equation can be considered strictly increasing and linear in this section of the complete motion.

## RAMP DOWN STAGE

This is the 3rd stage in the motion. This is duration when the turtlebot's speed starts reducing over time and it comes to a final resting position at the end. The logged Odometry messages are used to compute the ramp down time of the turtlebot. The end of the motion is marked as the timestamp when the move_base node publishes a "Reached the Destination" message. This message is printed on the console, since move_base is part of the launch file. The time when the ramp-down begins is marked by observing the instantaneous speed of the Turtlebot. This was done by marking the first instant at which the Turtlebot sees a more than 10% dip in it's instantaneous speed when travelling at the desired speed.

### CONSTANT SPEED MOTION

This is the section of the Turtlebot motion, when the robot moves at a constant velocity. More than 93% of the total distance is covered during this time. It's the time between the end of the ramp-up and the start of the ramp-down stage. During this period the Turtlebot moves at an instantaneous speed which is at a maximum of 10% deviation from the desired speed.

## DATA-COLLECTION

The data collection was accomplished by sampling the linear X direction speed from the /odom node at a rate of 30 Hz. The logs have been saved in the format: p<desired-speed>_<date>.log with the timestamp followed by the linear X speed. A total of 29 readings were done with speeds ranging from 0.2m/sec to 0.8m/sec with a step size of 0.05m/sec. The analysis was done for 2 sets of total distances: 8.1m and 15.1m. Additional experiments were done for speeds: 0.8m/sec, 0.65m/sec, 0.4m/sec and 0.45m/sec to have some random data points. The start point for the turtlebot was kept the same in all experiments: (X: 14.9m, Y: 16.9m). The destination was The instruction l1_to_l6.ig was modified to test for different distances in the X direction on the Wean-complete-floor4.world map.

```
P(V(1, do MoveAbsH(30.00, 69.00, 0.70, 0.0000) then 2),

V(2, end)::

nil)
```

### EXPERIMENTS

All the experiments were conducted on a CMU MARS platform on the ROS Indigo TurtleBot Simulator. A set of 29 runs of Turtlebot were conducted with varying desired linear velocity in X direction.

### ASSUMPTIONS

- For the purpose of modeling the linear motion of the Turtlebot, the instructions have only linear component in the x direction. The y velocity direction is maintained 0.
- There are no physical obstacles in the path for the Turtlebot to overcome.
- The y coordinate of the destination and the starting point is the same.
- Path has no inclination, is fairly flat and uniform throughout.
- There is no rotational component in the instructions given as an input for the motion experiments. The differences in the initial and final angular pose of the Turtlebot are managed by the move_base component and are accounted as is in the ramp-up and ramp-down periods.
- The motion equation estimates total time for distances greater than equal to distance that can be covered during the ramp-up and ramp-down motion and fails for distances less than that.
- If the distance is smaller than that, the model displays a warning and returns the total time as the sum of ramp-up and ramp-down time only. It also mentions the speed during the two sections of motion.

### SERIES OF STEPS FOLLOWED

With the CMU MARS Virtual Machine setup complete: A total of 29 readings were made to come up with the mode.

- Open a tab and run the ros launch file `./run-cp1`
- Change the desired speed and distance to be covered
    - cd ~/catkin_ws/src/cp_gazebo/instructions

- Open the l1_to_l6.ig and update the instruction graph to
  ```
  P(V(1, do MoveAbsH(19.80, 69.00, 0.35, 0.0000) then 2),
  V(2, end)::
  nil)
  ```
- Here 19.80 is the destination's X position, 69.00 is the destination's Y coordinate, 0.35 is the desired speed in X direction and the last parameter being 0 represents 0 Y-direction speed.

- On the first terminal wait for the message odom received which indicates that new tests can be run. The check on the status of the turtlebot and it's position by polling the observe service by running this command on a new terminal:
  - `localhost:5000/action/observe`
- If the observe service returns the position of the turtlebot, Start listening to the /odom topic to know the current position of the bot in the simulated environment. Use the sensorListener_shory.py to log the time and current speed.
  `./sensorListener_short.py > r<Speed>_<MMDD>.log`
- Start the run: `curl -H "Content-Type:application/json" -X POST localhost:5000/action/start`
- Observe the move_base messages printed on the first terminal. Note the time instants when the message about setting the speed to the desired speed is published. Also mark the instant when "Reached the destination" message is published. These indicates the start and end of the motion of Turtlebot.

# DATA ANALYSIS

## RAMP-UP ANALYSIS

The time taken to complete the ramp up motion was observed to vary directly with the desired speed. The plot below shows ramp-up time vs the desired speed.

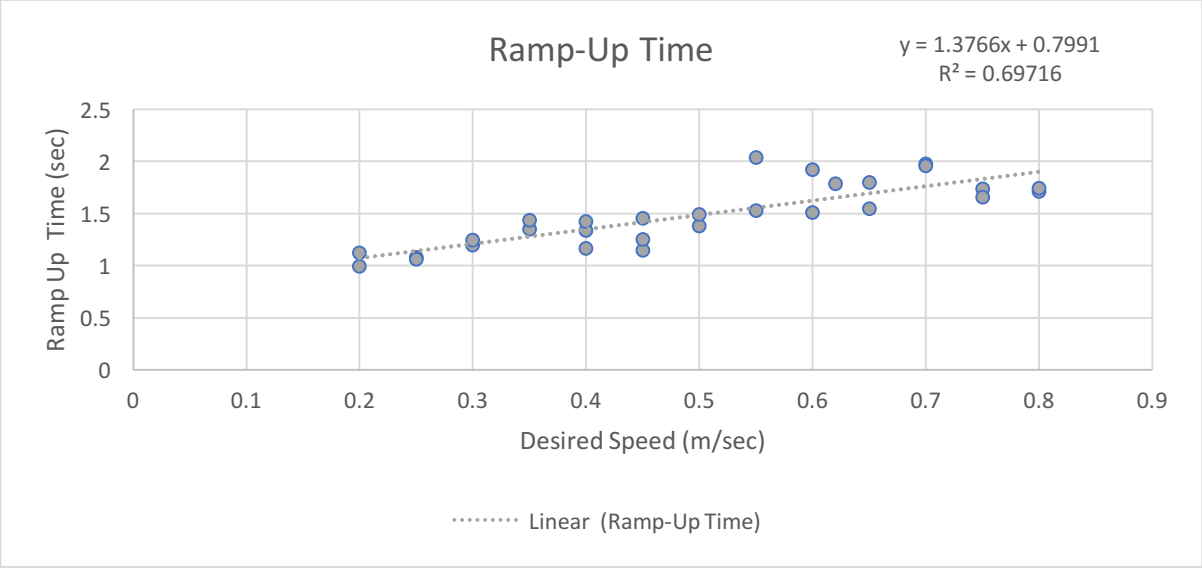


**FIGURE 2: RAMP-UP TIME VS DESIRED SPEED**

Linear Regression has been used to represent the relationship between desired speed and time spent in ramp-up.

RAMP-UP TIME DEPENDENCE ON DESIRED SPEED (S)

To see the dependence of the ramp-up time on the desired speed, the statistical significance was computed using t-tests, with the default Null Hypothesis as:

*Null Hypothesis*: The ramp-up time is independent of the desired speed (S) provided as an input.

For the t-tests, multiple runs (10) of the Turtlebot at 2 different speeds were conducted. The chosen S values being: 0.25 m/sec and 0.7m/sec since they lie at the opposite ends of the possible S values supported by the turtlebot. The t-test gave a p-value which can be used as a means to reject or accept the Null Hypothesis. The results of the tests are:

The two-tailed P value equals 0.1524, which is statistically significant to reject the Null Hypothesis and thus we can say that ramp-up time is dependent on the input desired speed (S).

| **Desired Speed (S)** | **Ramp-Up Time** |
|---|---|
| 0.7 | 1.5 |
| 0.7 | 1.504 |
| 0.7 | 1.849 |
| 0.7 | 1.53 |
| 0.7 | 1.743 |
| 0.25 | 1.25 |
| 0.25 | 1.289 |
| 0.25 | 1.629 |
| 0.25 | 1.562 |
| 0.25 | 1.557 |

| **Group** | **Group 1: (0.7m/s)** | **Group 2: (0.2m/s)** |
|---|---|---|
| Mean | 1.6252 | 1.4574 |
| SD | 0.16077 | 0.17441 |
| SEM | 0.0719 | 0.078 |
| N | 5 | 5 |
| | The two-tailed P value equals 0.1524 | |

**TABLE 1: T-TEST FOR RAMP-UP TIME DEPENDENCE ON DESIRED SPEED**

### RAMP-UP TIME DEPENDENCE ON TOTAL DISTANCE (D)

To look for the dependence of ramp-up time on the total distance, the technique of t-tests were used to find the statistical significance of the dependence. The Null Hypothesis in this case being: *The ramp-up time is independent of the total distance (D) provided as an input*. For this, the motion data for total distance equal to 8.1m and 15.1m was used. The desired speed was varied from 0.2m/sec to 0.75m/sec with step size of 0.05m/sec.

| **Distance** | **Ramp-Up Time** |
|---|---|
| 8.1 | 0.994 |
| 8.1 | 1.084 |

| | |
|---|---|
| 8.1 | 1.196 |
| 8.1 | 1.349 |
| 8.1 | 1.341 |
| 8.1 | 1.457 |
| 8.1 | 1.383 |
| 8.1 | 2.041 |
| 8.1 | 1.92 |
| 8.1 | 1.789 |
| 8.1 | 1.98 |
| 8.1 | 1.738 |
| 15.1 | 1.122 |
| 15.1 | 1.065 |
| 15.1 | 1.249 |
| 15.1 | 1.439 |
| 15.1 | 1.425 |
| 15.1 | 1.148 |
| 15.1 | 1.494 |
| 15.1 | 1.528 |
| 15.1 | 1.514 |
| 15.1 | 1.548 |
| 15.1 | 1.957 |
| 15.1 | 1.658 |

| **Group** | **Group 1: (8.1m)** | **Group 2: (15.1m)** |
|---|---|---|
| Mean | 1.52267 | 1.42892 |
| SD | 0.35912 | 0.25314 |
| SEM | 0.10367 | 0.07307 |
| N | 12 | 12 |
| | The two-tailed P value equals 0.4676 | |

**TABLE 2: T-TEST FOR RAMP-UP TIME DEPENDENCE ON TOTAL DISTANCE**

The t-test result shows that the P value is 0.467. This value is statistically not significant to reject the Null hypothesis about the ramp-up time independence from total distance input parameter.

From the above two t-test experiments using the two input parameters, desired speed and total distance, it can be concluded that only the former one plays a role in estimating the total motion duration. So we can use the desired speed dependence to come up with a linear regression based equation for the ramp-up time.

$$t_{rup} = 1.3766 * S + 0.7991$$

*Equation. 1*

The bias of 0.7991 handles any initial state parameters which may be responsible for the ramp-up time.

The logged data has also been used to calculate the average speed during the ramp-up time. This was achieved by calculating the average value of the instantaneous speed in this interval. The graph below shows the average speed during ramp-up vs the desired speed. The graph clearly shows an increase in average ramp-up speed for increasing desired speed (S) values.

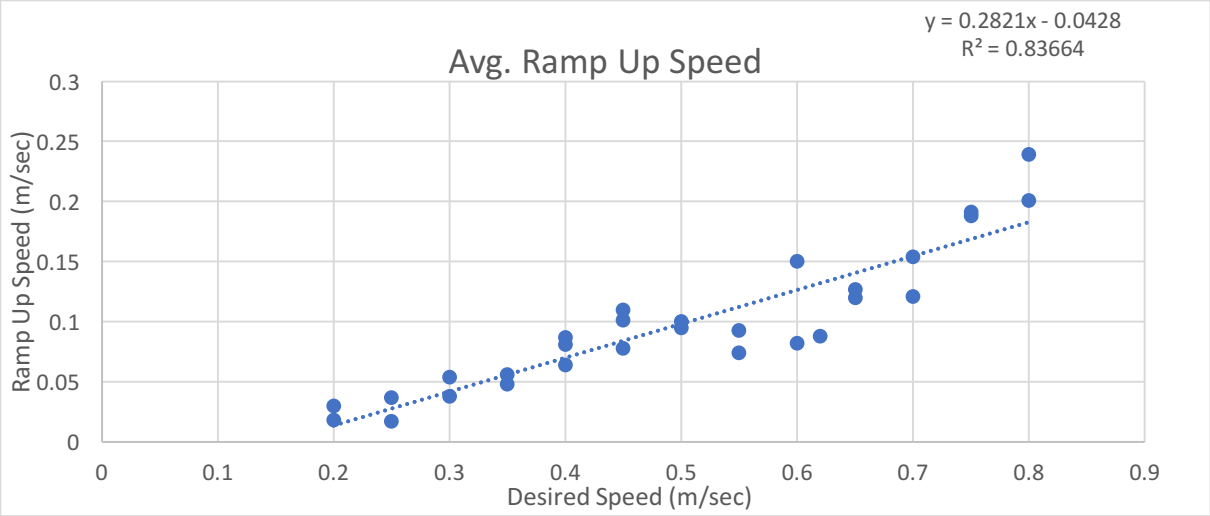


**FIGURE 3: RAMP DOWN SPEED VS DESIRED SPEED**

$$s_{rup} = 0.2821 * S - 0.0428$$

*Equation. 2*

## RAMP-DOWN ANALYSIS

Ramp down motion analysis showed a pretty different trend than that observed in the ramp-up time analysis. As can be seen from the below graph between ramp-down time vs desired speed, we see a non-linear pattern.

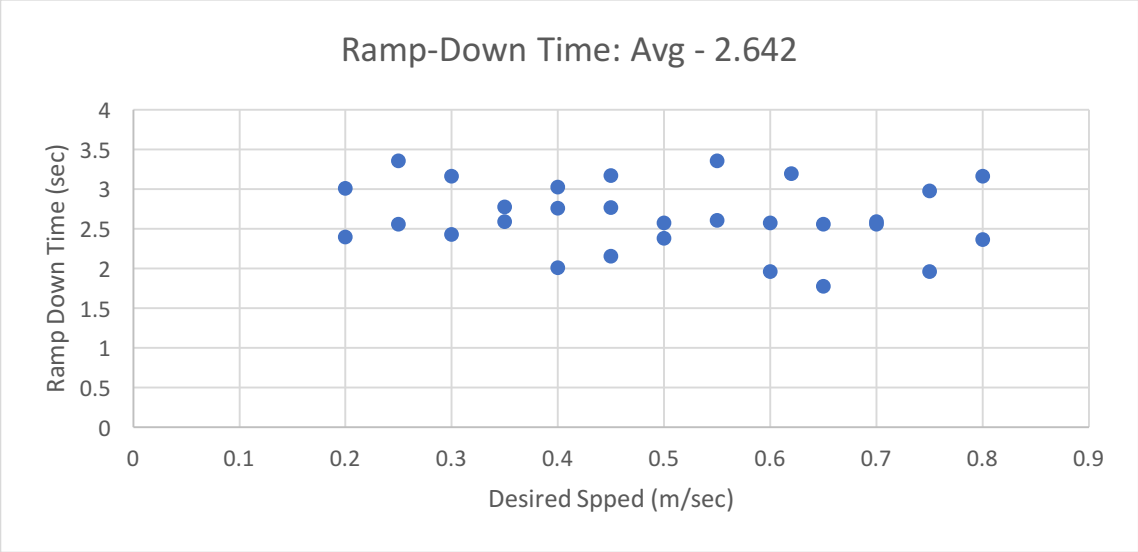


**FIGURE 4: RAMP-DOWN TIME VS DESIRED SPEED**

### USING DESIRED SPEED TO COMPUTE RAMP-DOWN TIME

To see the dependence of the ramp-down time on the desired speed, the statistical significance was computed using t-tests, with the default Null Hypothesis as:

*Null Hypothesis*: The ramp-down time is independent of the desired speed (S) provided as an input.

For the t-tests, multiple runs (10) of the Turtlebot at 2 different speeds were conducted. The chosen S values being: 0.25 m/sec and 0.7m/sec, since they lie at the opposite ends of the possible S values supported by the turtlebot. The t-test gave a p-value which can be used as a means to reject or accept the Null Hypothesis. The results of the tests are:

The two-tailed P value equals 0.6690, which is statistically not significant to reject the Null Hypothesis and thus we can say that ramp-down time is independent on the input desired speed (S).

This led to modelling the ramp-down speed as a fixed constant in the complete motion equation rather than being affected by the desired speed or total distance.

So ramp-down time for time model has been computed by using the average value for ramp-down time observed in the 29 experiments.

$$t_{rdown} = 2.642\ secs$$

*Equation. 3*

| Desired Speed (S) | Ramp-Down Time |
|---|---|
| 0.7 | 2.158 |
| 0.7 | 2.298 |
| 0.7 | 2.501 |
| 0.7 | 1.718 |
| 0.7 | 2.651 |
| 0.25 | 2.42 |
| 0.25 | 2.05 |
| 0.25 | 2.252 |
| 0.25 | 2.339 |
| 0.25 | 2.691 |

| Group | Group 1: (0.7m/s) | Group 2: (0.2m/s) |
|---|---|---|
| Mean | 2.2652 | 2.3504 |
| SD | 0.35932 | 0.235 |
| SEM | 0.16069 | 0.1051 |
| N | 5 | 5 |
| | The two-tailed P value equals 0.6690 | |

**TABLE 3: T-TEST FOR RAMP-DOWN TIME DEPENDENCE ON DESIRED SPEED**

The average speed during the ramp-down motion is calculated by averaging the instantaneous speed recorded during the ramp-down time. The average speed in the ramp-down segment vs desired speed in plotted below. The graph clearly shows an increase in average ramp-down speed for increasing desired speed (S) values.

Linear regression has been used to show the variance of avg. ramp-down speed with different desired speeds.

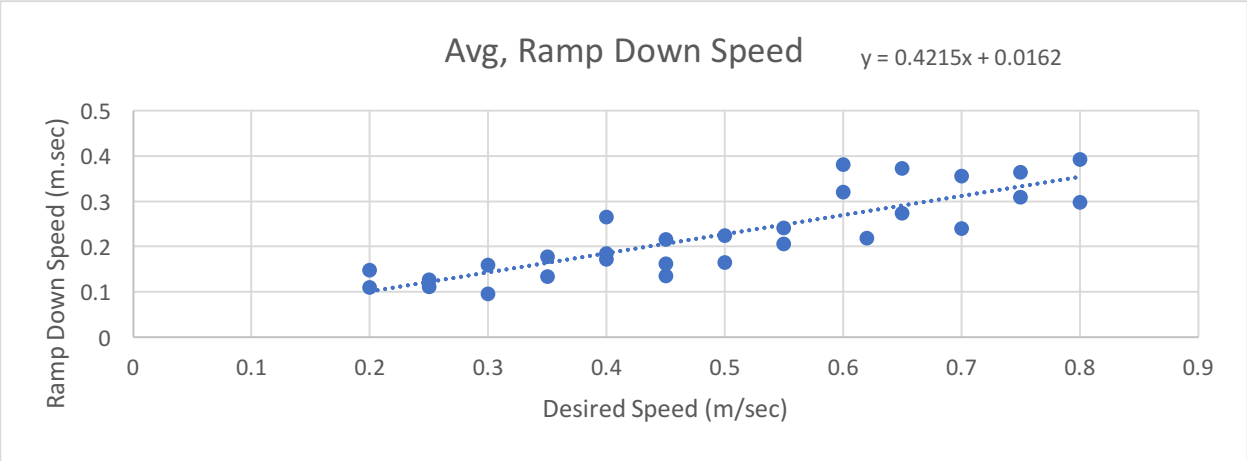


**FIGURE 5:RAMP-DOWN SPEED VS DESIRE SPEED**

$$s_{rdown} = 0.4215 * S + 0.0162$$

*Equation. 4*

### CONSTANT SPEED ANALYSIS

This is the time duration calculated by subtracting the total time by ramp-up and ramp-down time. The average speed in this duration is equal to the desired speed. The distance travelled at this speed can be computed subtracting total distance by distance travelled in ramp-up and ramp-down motion.

$$d_{const} = D - d_{rup} - d_{rdown}$$

*Equation. 5*

## TIMING MODEL

### INPUTS

S: desired speed

D: total distance to be covered by the Turtlebot

### OUTPUT

T: total time taken predicted by the model

$$T = t_{rup} + t_{const} + t_{rdown}$$

*Equation. 6*

$t_{up}$: Ramp-Up time. Proportional to desired speed S.

$t_{down}$: Ramp-Down time. A constant value.

$$T = t_{rup} + \frac{D - d_{rup} - d_{rdown}}{S} + t_{rdown}$$

*Equation. 7*

$d_{up}$: Ramp-Up distance.

$d_{down}$: Ramp-Down distance.

$$\boldsymbol{T = t_{rup} + \frac{(D - t_{rup} * s_{rup} - t_{rdown} * s_{rdown})}{S} + t_{rdown}}$$

*Equation. 8*

$s_{up}$: Ramp-Up average speed. Proportional to desired speed S.

$s_{down}$: Ramp-Down average speed. Proportional to desired speed S.

## VALIDATION OF THE TIMING MODEL

Validating the model proposed above in Equation 8, is a key thing to show that it holds true for future cases and is useful at motion prediction.

To do this, a cross validation technique called Leave one out Cross Validation was deployed on the entire data set. A timing model was developed using all but one row in the data table. The model was then used to compute the predicted time for completion of the motion for the row left out above. The error was calculated by comparing the total time observed and the time predicted by the Leave one out Cross Validation model. This is called the training error.

The average training error by using the prediction model was noted to be ***1.85***% of the observed time for the complete motion.

$$Absolute\ Training\ Error\ (e) = |\ (Observed\ Total\ Time - Predicted\ Total\ Time)\ |$$

$$\%Training\ Error = \frac{Absolute\ Training\ Error\ (e)}{Observed\ Total\ Time}$$

*Equation. 9*

## ANNOTATED BIBLIOGRAPHY

[1] http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/06-motion-models.pdf : Odometry based vs Velocity based motion modelling. Representation of robot motion in 2D plane and discussion on how to account for angular and linear motion.

[2] https://en.wikipedia.org/wiki/Linear_encoder: Basics on encoders

[3] http://wiki.ros.org/: Details on how to use the ROS middleware, Writing subscribers, packages like move_base and navigation packages.

[4] https://www.graphpad.com/quickcalcs/ttest1/: Used to do the t-test on the data collected. Get the p-value.

[5] https://www.investopedia.com/terms/t/t-test.asp: Significance of t-tests and how to come with Null Hypothesis.

[6] https://lagunita.stanford.edu/c4x/HumanitiesScience/StatLearning/asset/cv_boot.pdf: Leave-one-out-cross-validation technique.

# ATTACHMENTS

## CODE

- sensorListener.py: Subscriber to /odom messages for linear x direction velocity
- parser.py: parse log file for only timestamp and linear x velocity
    - `python parser.py <log_file> > <output_file_name>`
- turtlebot_timeEstimation.py : Estimate total time for desired speed and total distance
    - `python turtlebot_timeEstimation.py -s <desired_speed> -d <total distance>`

### DATA COLLECTION

- Motion_Data.xls
    - Data Tab: All experiments, readings and calculations for times, velocity and distance
    - Plots Tab: Time and speed plots
    - Validation: Model validation and error calculation
    - T-tests: Data on t-tests using one high and one low speed to statistically vouch for dependence of ramp-up time on desired speed and no such dependence for ramp-down time.
- RunsData.zip: The logs for data collection of 29 runs used to develop the motion model.