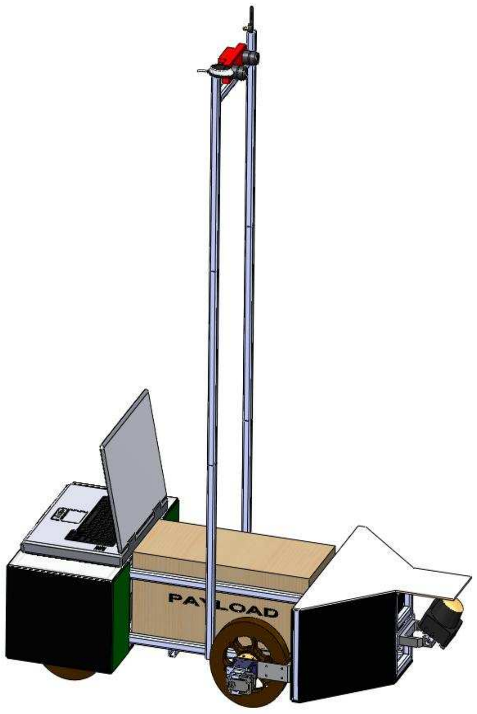
NATIONAL UNIVERSITY OF SINGAPORE

In collaboration with

A*STAR



[DESIGN REPORT : LEO10]

Design Report IGVC 2010
LEO10

National University of Singapore's Entry to IGVC 2010

Dev Chandan Behera, Ankit Sachdev, Hitesh Dhiman, Dr. Prahlad Vadakkepat
National University of Singapore

***Abstract*— We present LEO10 an autonomous ground vehicle developed by the KARR team from the National University of Singapore (NUS). LEO10 is a lightweight, modular, power efficient intelligent autonomous robot running on Robot Operating System (ROS). This robot is National University of Singapore's first entry at the Intelligent Ground Vehicle Competition (IGVC) in collaboration with Data Storage Institute, A*STAR. LEO10 is a completely in-house designed robot from scratch by three undergraduate students.**

***Keywords**- ligtweight, modular, power efficient, ROS, IGVC*

I. INTRODUCTION

The team members of KARR got together in early August 2009 to brainstorm ideas on how to design and create a vehicle that will stand out at the competition. We used a systems based design approach to build the autonomous vehicle from scratch.

The software platform we will be using for the robot is Robot Operating System (ROS) developed by Willow Garage Inc. which will run on Ubuntu and Arduino for our motor control.

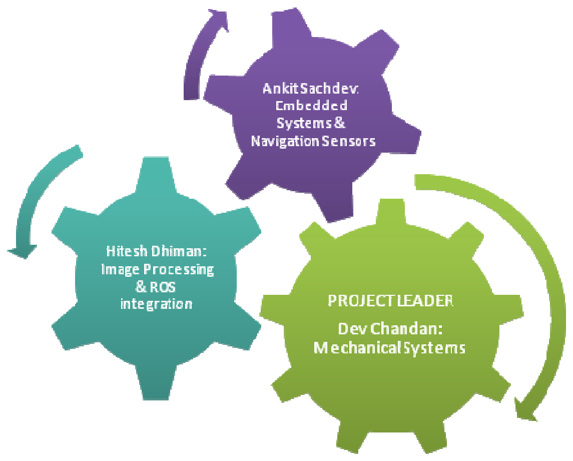
After thorough research from previous year's entries at IGVC, it was decided to build a lightweight, modular and low power consuming robot for IGVC. The advantages that Robot Operating System (ROS) offered over Stage and LabView confirmed ROS as the software running on the robot with Arduino to be used for Motor Control.

The objectives of the Robot were finalized that led to the subsequent stage of selection of components and LEO10 was successfully fabricated completely in-house from scratch in 1 semester to compete in IGVC.

II. DESIGN PROCESS- ITERATIVE DEVELOPMENT

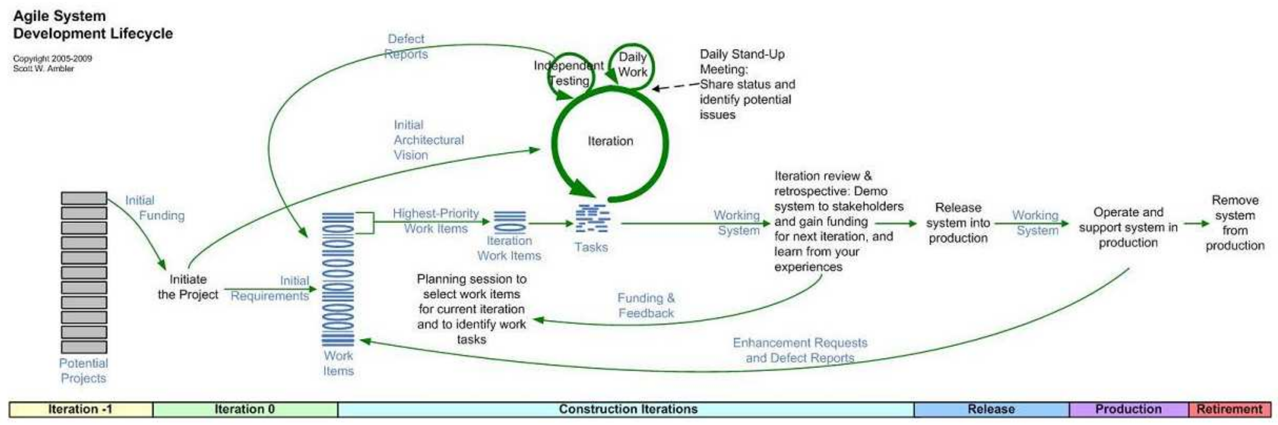
The agile iterative development process was used to start the KARR team and build the robot, LEO10 as an effort to start an undergraduate student IGVC team in NUS and to ensure continuity of the team.

Figure 1. The Team


Ankit Sachdev: Embedded Systems & Navigation Sensors
Hitesh Dhiman: Image Processing & ROS integration
PROJECT LEADER
Dev Chandan: Mechanical Systems

We brainstormed over the various possible features that could be incorporated in Leo10 along with the basic capabilities required to compete at IGVC. The priority level of all the features was evaluated after which the list was cut down. The tasks were thus divided into 3 segments with each individual responsible for the development of a particular set of features for LEO10. The components were chosen to suit our needs for the competition after ensuring their compatibility and performance with ROS. The challenges were prioritized based on the sensory requirements and the project was planned in the direction of completing the specific milestones at specific deadline. The software was developed and tested on the Pioneer P3DX before it was migrated to our own robot, LEO10.

Figure 2. Agile Development Process


Agile System Development Lifecycle
Copyright 2005-2009 Scott W. Ambler
Defect Reports
Independent Testing
Daily Work
Daily Stand-Up Meeting: Share status and identify potential issues
Iteration
Initial Architectural Vision
Initial Funding
Highest-Priority Work Items
Iteration Work Items
Tasks
Working System
Iteration review & retrospective: Demo system to stakeholders and gain funding for next iteration, and learn from your experiences
Release system into production
Working System
Operate and support system in production
Remove system from production
Initiate the Project
Initial Requirements
Planning session to select work items for current iteration and to identify work tasks
Funding & Feedback
Enhancement Requests and Defect Reports
Potential Projects
Work Items
Iteration -1
Iteration 0
Construction Iterations
Release
Production
Retirement

III. Design Objectives

The task of building the robot from scratch required clear design goals which were:

A. *Lightweight:*

To be able to move smoothly, decrease the work done by the motor and reduce the power consumed by the drivetrains, it was essential to decrease the weight of the robot. We prepared a weight budget of the mechanical structure, drivetrains and other hardware components to optimise the weight of the robot to 23 kg which would be considered as one of the lightest vehicle at the competition. The drive trains and frame structure will be covered in detail in the mechanical design section

B. *Compact:*

The idea behind making a compact and modular robot was the ease of transportation and storage. Moreover, modularity of the structure offers the possibility of customizing the dimensions of the robot subjected to the hardware or task requirements of the end users.

C. *Power Miser*

The design process also included the power budget of the robot which we think is an essential part of the process in the design as the robot should have a good amount of battery life to be autonomously running.

IV. LEO10 System Overview

From the point the robot is switched on the three sensors i.e. LIDAR, Camera and the IMU/GPS system start to poll data simultaneously. The navigation algorithm in ROS takes this information and plans the path accordingly. In the autonomous challenge the robot will be continuously given pose information as its goal to move forward where as in the Navigation challenge the waypoints will determine the robots goal. Figure 3 shows the complete system diagram of LEO10.

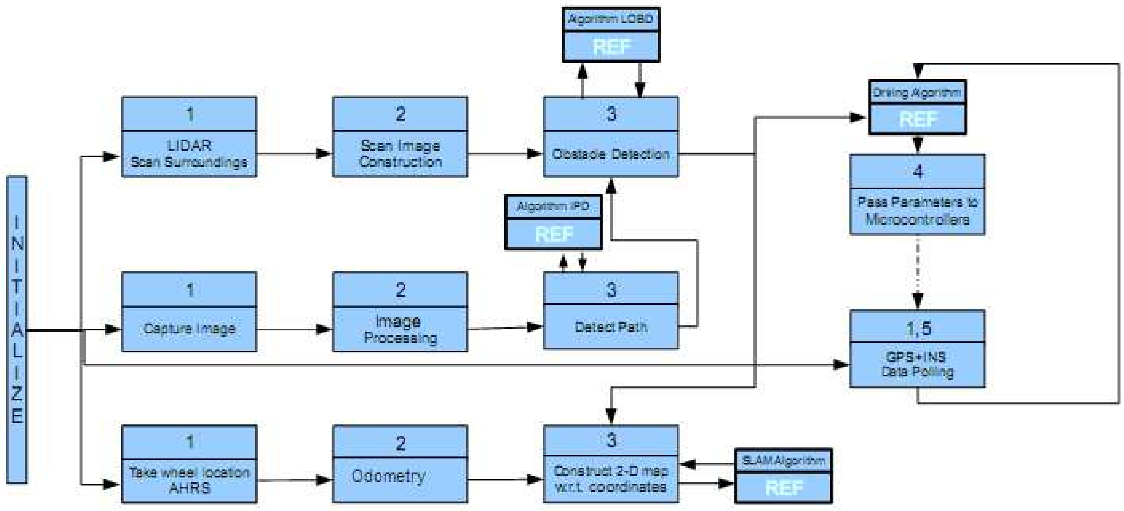


Figure 3. System overview

V. Mechanical Systems

Apart from being lightweight and compact one important feature LEO10's mechanical design is that all the drivetrains are independent of each other and is a four Wheel Skid Steer Mechanism. Independent drive trains with individual suspension that will add to the all terrain capability of the robot. However for this year, we decided to go without suspension for simplicity of the first design of the robot.

TABLE I. Mechanical DataSheet

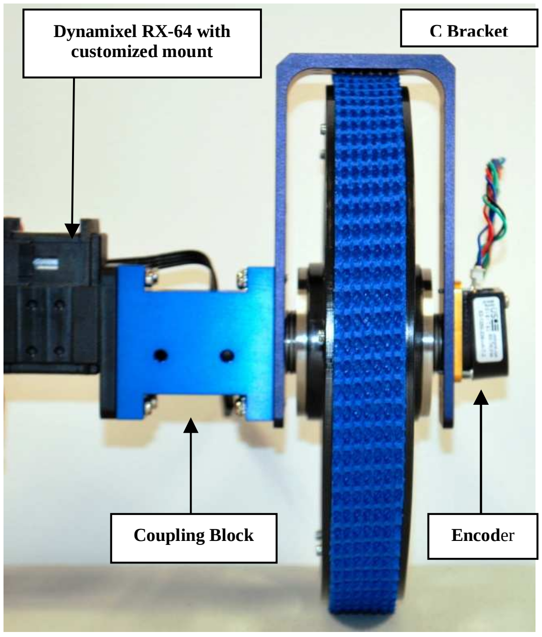
Length	**1.10 m**
Width	**0.64 m**
Height	**1.49 m**
Weight(without payload)	**23 kg**
Ground Clearance	**0.05m**
Payload Capacity	**20 kg**
Wheel Diameter	**8 inches**
Maximum Speed(80% Efficiency)	**1.85 m/s**
Operational Speed	**5 m/s**

A. *Methods of Design*

LEO10 was conceptualized and designed in detail in Solidworks 2009. Each component and assemblies were confirmed as non-interfering and constrained using the computer aided design (CAD) program. Simulation Study features in the program was used to do the Finite Element Analysis (FEA) so as to ensure that the structural parts maintain a factor of safety of at least 5 under worse cases of loading.

B. DriveTrain Design

Leo10 has four independent drive trains each powered by light weight Dynamixel-RX 64 motors. The main feature about this design is the modularity of drive trains which makes replacement of parts and assembly of the design simpler. There is a motor plate customized to hold the Dynamixel motor which mounts onto the coupling block on which the weight of the robot actually acts. The Oldham coupling inside the block ensures transmission of power from the motor shaft to the wheel shaft. The wheel shaft is well supported by the flanged ball bearing on the C bracket on the other end of which the encoder plate housing the US Digital E2 Optical encoder is housed.

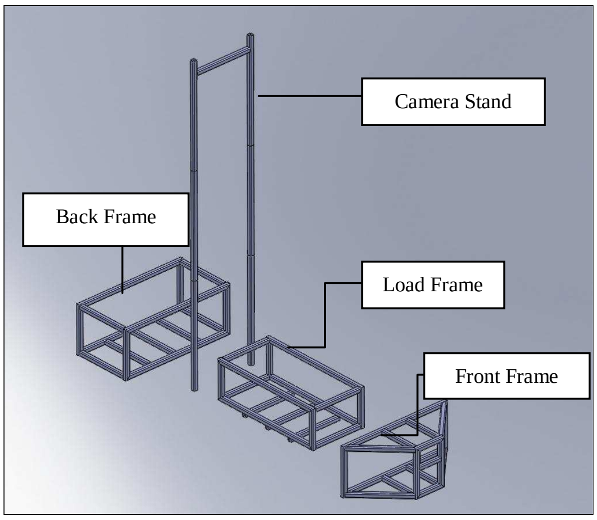


The power requirements for motors were calculated taking the scenario that the vehicle is climbing the ramp at an angle of 15°. The power required for each motor is 24.2 W.

All the parts were fabricated from AL-6061-T6 aluminum except for the shafts which were fabricated from 403 Stainless Steel.

C. Profile Body Design

The chassis was fabricated from Item® profiles 20 mm by 20 mm in dimension. These profiles are light weight and provide the option of mounting at desired points. Hence it is easier to adjust the heights of the LIDAR and the camera when mounted on these profiles.



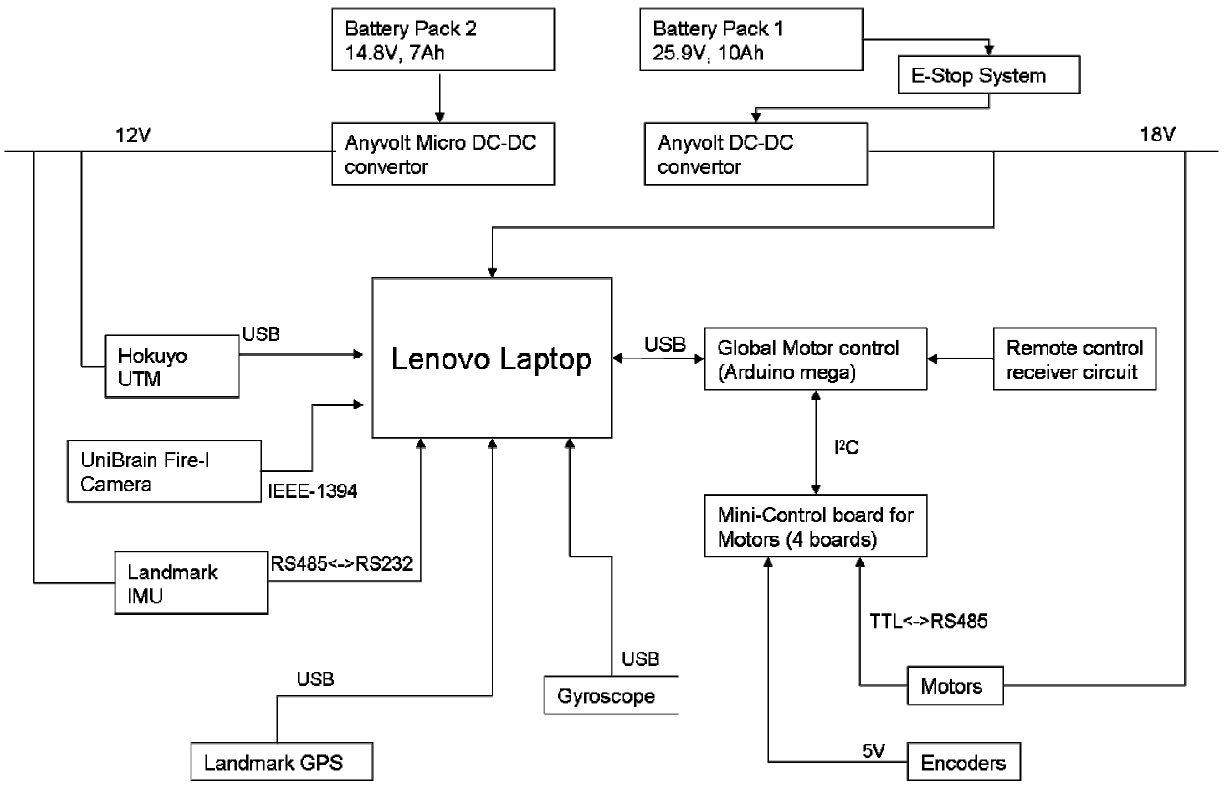
The weight distribution was done equally in the front and the back frame and the drive trains were mounted at a ground clearance of 5 cm. The centre of gravity of vehicle is towards the centre of the robot and low. The total weight of the chassis is about 7 kg.

D. Enclosure and Sensor Mounts

The front and the back frame were enclosed with scratch resistant plastic panels sealed with rubber lining on the profiles. All the circuits and Unibrain camera are protected inside IP66 containers to prevent water from damaging the components. The laptop will be placed in a safe enclosure that will mount onto the back frame. For easy access of the components inside the back frame is a hinged panel.

The Hokuyo LIDAR is mounted to Dynamixel AX-12+ motor to provide 3D scanning. It is protected by the protrusion of the panel on top of it.

VI. ELECTRONICS SYSTEM



A. *Wireless & Hardware Emergency Stop.*

LEO10 is installed with a wireless relay board with an Arduino-Mini Pro microcontroller and an AR6200 DSM2 receiver to enable remote emergency stop capability. Spektrum DX6i receiver has 6 different RF channels, amongst which the gear channel is monitored for the emergency stop (figure 1.1); since it is has only two states. The wireless receiver relay board is connected in series with the main relay power board to stop the current flow to the motors when the e-stop button is pressed.

For the hardware E-Stop, a push button is placed on the back of the robot connected to the main power board, which shuts off all power/electronic operations on the robot. Unwinding the pushbutton will reset the robots systems and return to normal state.

B. *Manual Control*

The AR6200 DSM2 receiver, used for the wireless emergency stop will also be used to manually drive the robots with a joystick. The AR6200 DSM2 has 6 different channels which are usually used as receivers on RC aircrafts. In our case we are going to use only 2 channels to drive the robot. The rudder and throttle joystick on the DX6i transmitter is used to control the direction and the speed of the robot, respectively. An Arduino-Mini Pro microcontroller is used to process the PWM signals from the receiver and output the desired controls to the global motor control board.

C. *Drivetrain Control System*

Each of the 4 drivetrains on the robot is equipped with Dynamixel's RX64 motors, which can handle 6.4 N/m with at a maximum RPM of 114 at max operating voltage. The motors have an internal controller and driver which can be controlled by packet communication through a bus, supporting RS485 network.

Our motor control system consists of 2 layers, 1 global control board and 4 custom mini-boards for each drive train. The global motor controller is an Arduino Mega board with an Atmel 1280 chip, which is programmed to manage the communication between the path planning algorithm on the computer and each motor control board as well has handle the link between the wireless transmitter and receiver for manual driving.

Each mini control board consists of an Arduino mini, RS 485 breakout chip and other power modules. The micro-controller gets speed values from the Global motor control board and converts it into RX-64 communication packets to drive the motor. It takes in encoder values, processes the data and sends it to the global motor control to carry out the kinematic modelling for Dead-Reckoning. Communication between the boards is handled by I2C. Each board also carries its own power regulation modules.

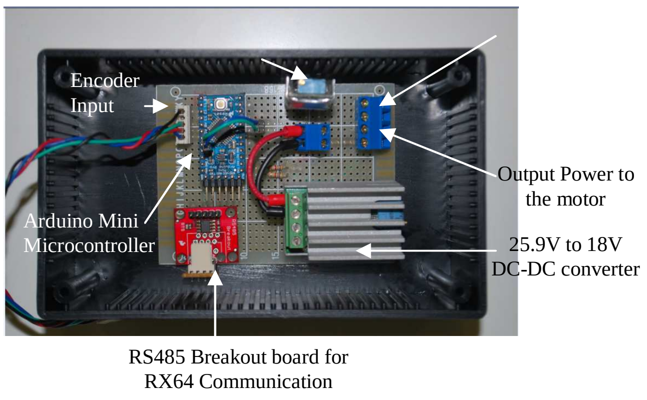

Encoder Input
Arduino Mini Microcontroller
Output Power to the motor
25.9V to 18V DC-DC converter
RS485 Breakout board for RX64 Communication

Figure 4. Circuit board inside the IP-66 casing.

VII. POWER SYSTEMS

A. *Power Budget*

One of our design objectives was to reduce the overall power consumption. Literary research suggested that drivetrains consume up to 60% of a robot's power capacity. With our innovative drivetrain design we have reduced the overall power consumption to 39% and still retained a sufficient maximum torque limit to traverse challenging terrains. The total power budget is given in the table

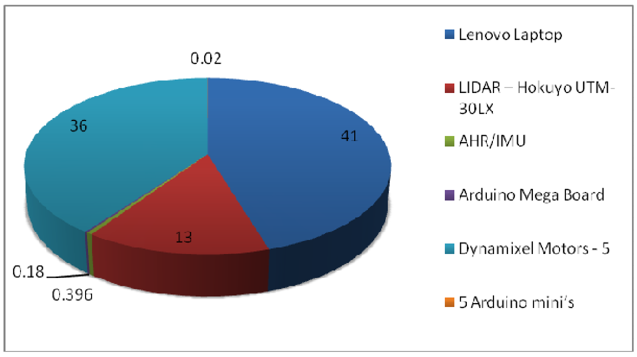


Figure Pie Chart for Power Consumption in Watts

The total power requirement of 90.59W is at optimized operation of the robot running at the speed of 0.5m/s. This makes our robot one of the most power efficient designs in the competition.

B. *Power regulation and distribution*

To go with the design objective of a light weight robot high power Li-Po batteries of 25.9V 10 Ah and 14.8V, 7Ah were used at a weight budget of 2.5kgs. Total Battery Run time for Battery pack is 2h 38 minutes at 25.9V and 6h 14 minutes at 14.8V. The battery voltages are stepped down to 18V, 12V and 5V respectively by Anyvolt 3 DC-DC converters and 25W Step down adjustable switching regulators.

C. *Battery Management*

Depleted batteries can be replaced easily without affecting the robots state as the laptop has its own internal battery and thus the software process keep running.

VIII. COMPUTING SYSTEMS:

A. *Software:*

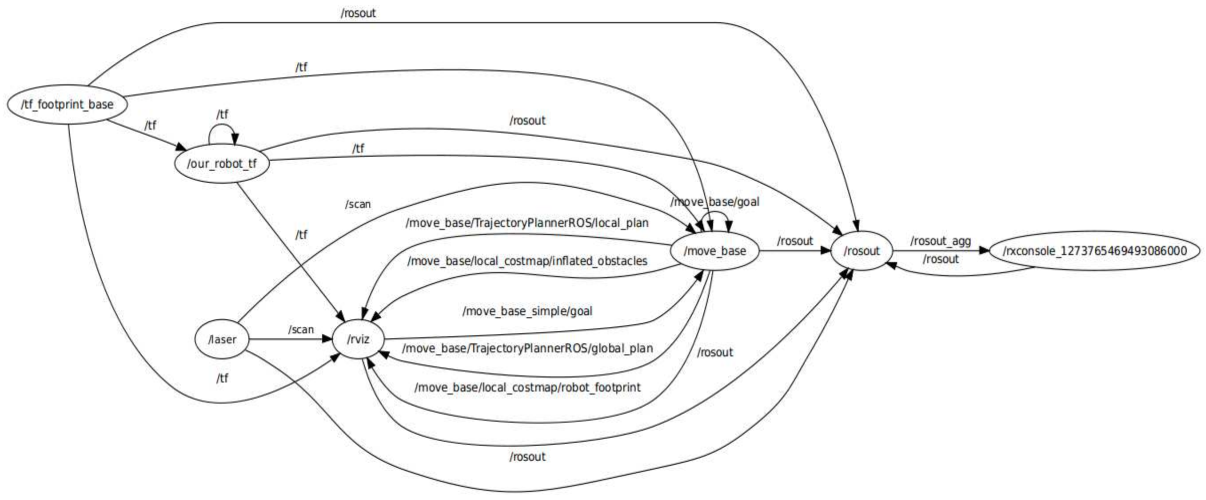
LEO10 uses Robot Operating System, or ROS as its software platform. ROS is an open source, 'meta operating system', and runs on Linux.

The basic motivation behind using ROS was:

- Open Source, hence lesser costs for software
- Stable and robust platform.
- Provides integration of OpenCV, Player/Stage and Gazebo under one package
- Well supported libraries for common algorithms.
- Supports both C++ and Python code.

As stated above, ROS behaves like a meta-operating system, where the 'core' provides the basis for all other 'packages' to run on it. The core handles all the communication processes, while packages are user written programs that run as separate 'nodes', connected to each other, forming a network.

The nodes running the navigation algorithm on our robot are given in figure 5.



/rosout
/tf
/tf_footprint_base
/tf
/tf
/rosout
/our_robot_tf
/tf
/scan
/move_base/goal
/move_base/TrajectoryPlannerROS/local_plan
/move_base
/rosout
/rosout
/rosout_agg
/rosout
/rxconsole_1273765469493086000
/move_base/local_costmap/inflated_obstacles
/tf
/move_base_simple/goal
/laser
/scan
/rviz
/move_base/TrajectoryPlannerROS/global_plan
/rosout
/tf
/move_base/local_costmap/robot_footprint
/rosout

Figure 5. ROS Nodes Overview

The encircled names are the 'nodes' and the arrows indicate the message names along with the direction of communication that these nodes use to compute tasks.

B. Lane Detection

Lane detection is done via a fire-wire camera (Unibrain Fire-i Board Pro).

ROS uses the cameradc1394 node to initialize the camera, and publishes the images under the /Image message. For image processing, we use OpenCV which takes in processed images in the ipl format, done by the CvBridge node in ROS.

Rather than relying on the visual information independently, the image processing is incorporated into the decision making process. This is done by converting the lines detected into laser scans, in a format that can be understood by ROS as a laser sensor. This means that the lanes detected appear as solid walls i.e. obstacles to the robot, and so the robot will try to avoid the side lanes.

C. Pose Estimation

For estimating the pose of the robot, we use EKF (Extended Kalman Filter). This feature has been implemented as the robot_pose_ekf node in ROS. This node takes in combined information from various sensors:

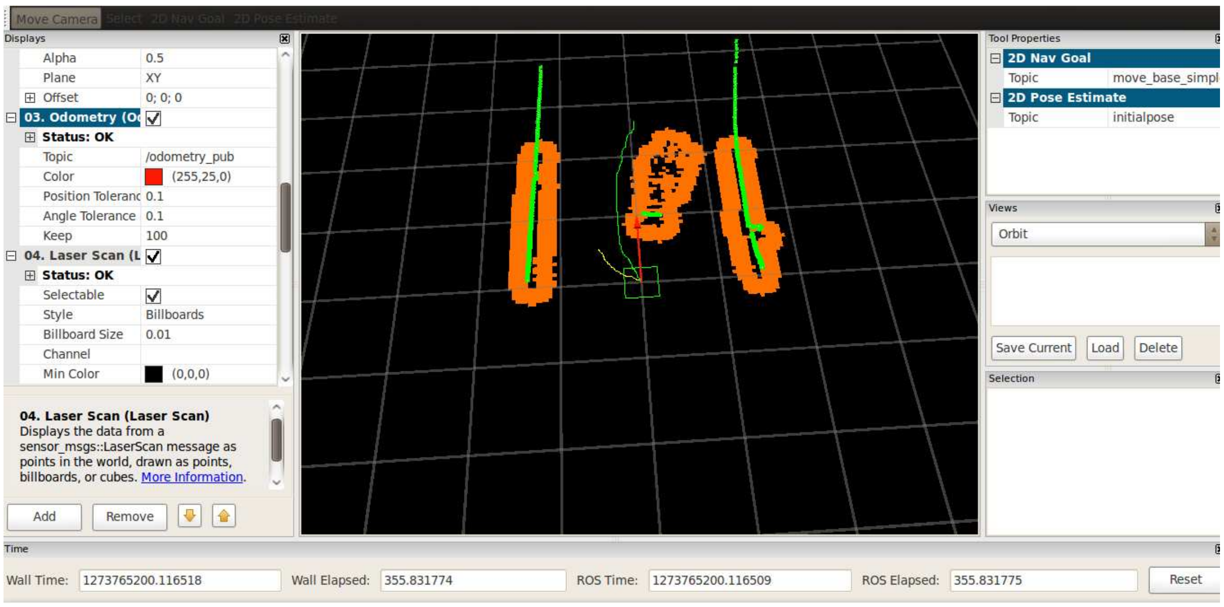
- 2D Pose: Wheel odometry gives the ground pose of the robot.
- 3D Pose: An IMU records information about the roll, pitch and yaw of the robot.
- 3D Position: A GPS sensor gives information on the position of the robot in the 3D plane.

D. Obstacle Avoidance & Navigation

Avoiding obstacles is the fundamental task of LEO10, is achieved using a Costmap based algorithm. ROS already has an implementation of the Costmap2D algorithm. Briefly, the algorithm can be explained as follows:

- Upon receiving real world obstacle information, the algorithm builds a 2D occupancy grid, where each cell that is known to have an obstacle is associated with a cost. The cost is maximum for obstacles near the robot (known as ‘inflated’ cost), and reduces with distance. Cells can be marked as ‘occupied’, ‘free’ or ‘unknown’ based on the information available about them and the costs they acquire. A total of 255 values are used for costs, grouped under the 3 categories stated above. Upon performing an initial run, a global path is planned, based on the lowest cost path available at that moment.
- As the robot moves along the path, there is a local plan that will iterate through the sensor information and plan velocity commands to drive the robot locally. The significance of the local path lies in its property; the global path can be planned with relatively inadequate information due to distance limitations of sensors (LIDAR for example). The local path works as a corrective measure: as the robot acquires information while moving close to the goal, the local path will adjust itself, avoiding obstacles while staying close to the global path. An example is shown in the simulation discussed in the following section.
- In cases where a path is not viable, or the robot is blocked, a 'recovery' operation is performed. This means that the robot will make a full turn of

360 degrees, and try to clear out its space by first maintaining a clear distance from any obstacles it is too close to, and then planning a path again. In Figure 6, the global path is shown in green, while the local path is shown in yellow. The orange area represents the inflated regions that have been assigned a cost. The red arrow marks the pose of the robot, and will travel with the rectangular 'footprint' of the robot.



Alpha	0.5
Plane	XY
⊞ Offset	0; 0; 0
⊟ 03. Odometry (Od	✓
⊞ Status: OK	
Topic	/odometry_pub
Color	🟥 (255,25,0)
Position Toleranc	0.1
Angle Tolerance	0.1
Keep	100
⊟ 04. Laser Scan (L	✓
⊞ Status: OK	
Selectable	✓
Style	Billboards
Billboard Size	0.01
Channel	
Min Color	⬛ (0,0,0)

04. Laser Scan (Laser Scan)
Displays the data from a sensor_msgs::LaserScan message as points in the world, drawn as points, billboards, or cubes. More Information.

⊟ 2D Nav Goal	
Topic	move_base_simpl
⊟ 2D Pose Estimate	
Topic	initialpose

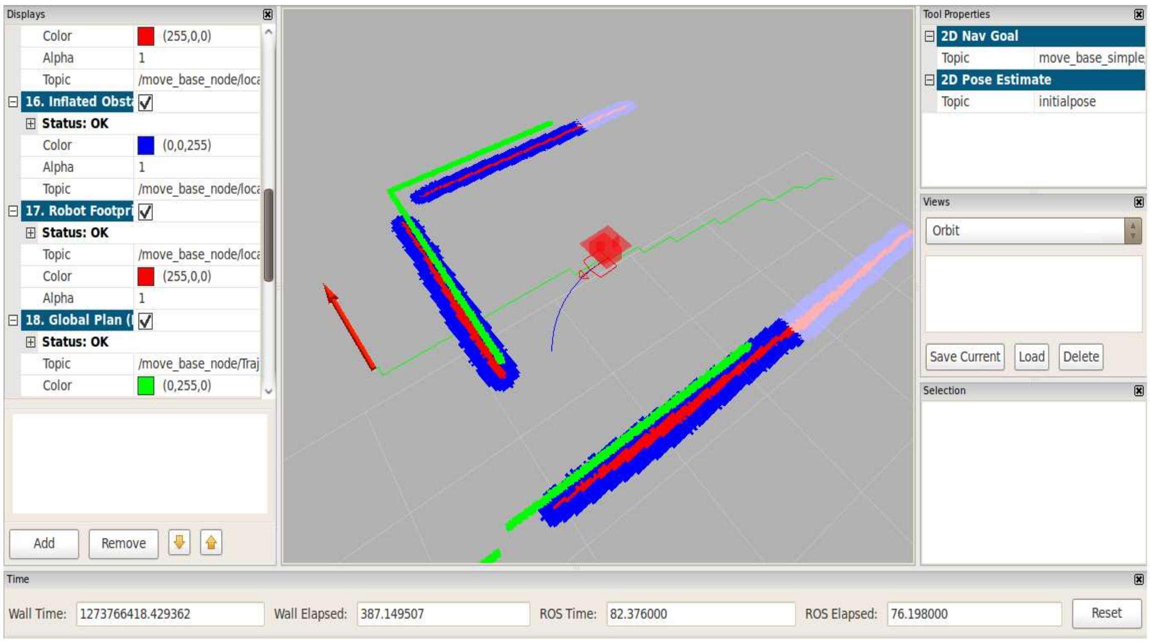
Orbit

Wall Time: 1273765200.116518 Wall Elapsed: 355.831774 ROS Time: 1273765200.116509 ROS Elapsed: 355.831775

Figure 6. Obstacle Avoidance Solution. Orange areas represent the inflation radius of the obstacles, the thick green lines represent the while lanes.

D. *Simulation*

To test the algorithm, we used the gazebo implementation in ROS. The robot was provided with a map, and then given a goal to achieve in the map. Examples are shown in the pictures below:


Displays

Color	🟥 (255,0,0)
Alpha	1
Topic	/move_base_node/loca
16. Inflated Obst:	✓
Status: OK	
Color	🟦 (0,0,255)
Alpha	1
Topic	/move_base_node/loca
17. Robot Footpr	✓
Status: OK	
Topic	/move_base_node/loca
Color	🟥 (255,0,0)
Alpha	1
18. Global Plan (	✓
Status: OK	
Topic	/move_base_node/Traj
Color	🟩 (0,255,0)

Tool Properties

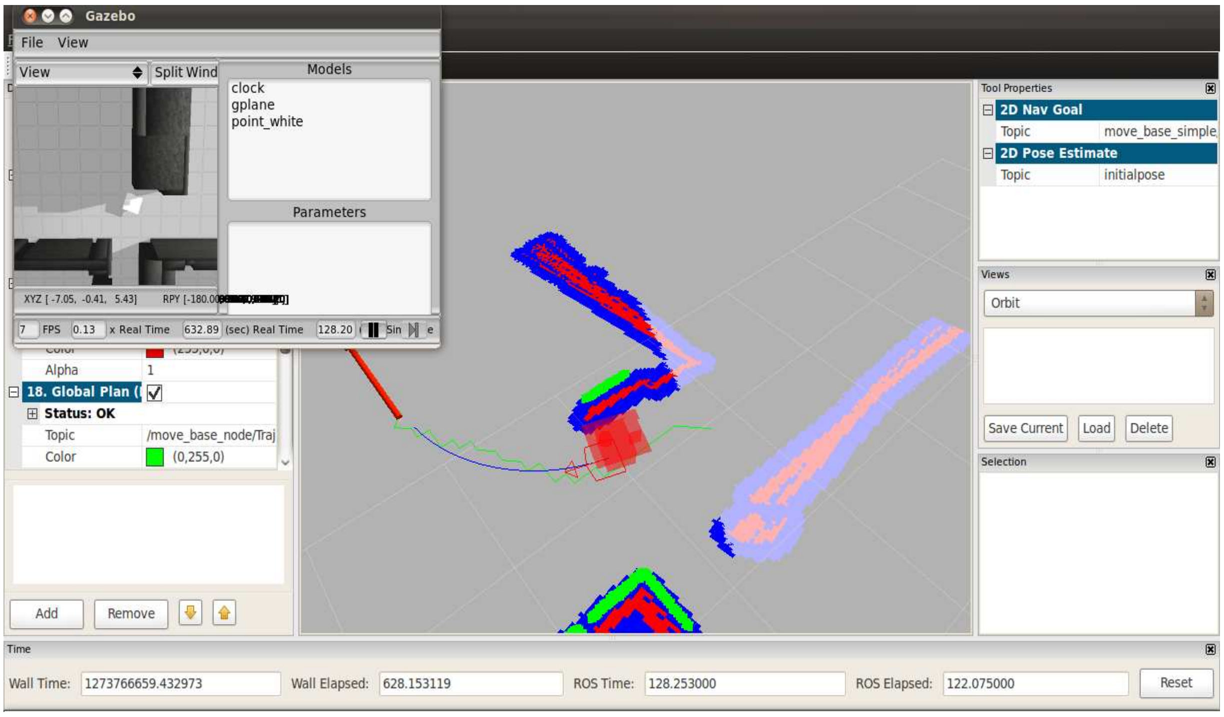
2D Nav Goal	
Topic	move_base_simple
2D Pose Estimate	
Topic	initialpose

Views: Orbit

Time

Wall Time: 1273766418.429362 Wall Elapsed: 387.149507 ROS Time: 82.376000 ROS Elapsed: 76.198000

Figure 7. Simulation Environment.


Gazebo

Models: clock, gplane, point_white

Parameters

XYZ [-7.05, -0.41, 5.43] RPY [-180.0

7 FPS 0.13 x Real Time 632.89 (sec) Real Time 128.20

Alpha	1
18. Global Plan (	✓
Status: OK	
Topic	/move_base_node/Traj
Color	🟩 (0,255,0)

Tool Properties

2D Nav Goal	
Topic	move_base_simple
2D Pose Estimate	
Topic	initialpose

Views: Orbit

Time

Wall Time: 1273766659.432973 Wall Elapsed: 628.153119 ROS Time: 128.253000 ROS Elapsed: 122.075000

Figure 8. Simulation Environment

Referring to figure 7 the thick green lines represent the walls detected by the robot, and the blue regions represent the costs assigned to these walls. The global plan passes through the wall because information about the front wall is not known at the starting point of the simulation. On reaching close to the wall, the local plan curves around the wall.

In the case of figure 8, the robot has a clear way to the goal, and thus the local plan coincides with the global plan.

IX. Control Systems & Sensors

The software development cycle involved a bottom up approach, i.e. design of the motor control was done first and the path-planning algorithms were implemented at the end of the development cycle. Each control system is described below in detail:

A. *Motor Control System*

1) *Mini-Control Board :*

Arduino mini pro microcontroller is programmed to perform the PID control of the Dynamixel motors. Since the Dynamixel has its own controller the ramping up of to the desired setpoint is done smoothly and the arduio's job is to maintain the output through feedback from the encoder values. The flowchart below shows the control algorithm implemented. *GMC = Global Motor Control board.

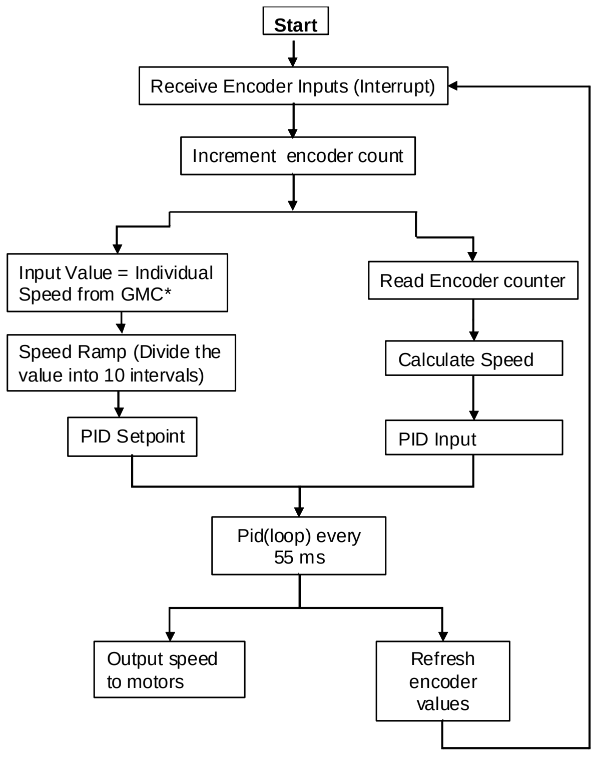


Figure 9. Flowchart of theMini-control board

2) *Global Motor control:*

GMC consists of only an 8 bit Arduino Mega controller which is responsible for the following tasks:

- Handling encoder information and sending the odometry values to the computer. Encoder values requested at a frequency of 50Hz.
- Communicates with the laptop to get the speed values for individual motors and passes the information over the I^2C channel, where all the motor control boards are connected to the same bus.
- When set to manual control mode, it receives speed values from the Wireless control system and performs inverse kinematics to set individual motor speeds for movement.

B. *LIDAR System – Obstacle and Ramp Detection*

Fixed single line LIDAR's are unable to differentiate obstacles from elevated paths such as ramps. To overcome this limitation LEO10 is installed with a tilting LIDAR mount , controlled by an AX-12 motor which rotates the Hokuyo UTM-30 LX (30m, 270° scanning range) in the Y axis allowing it to take multiple scans, and generate a 3D point cloud. We then use ROS's point cloud library to distinguish obstacles from the path.

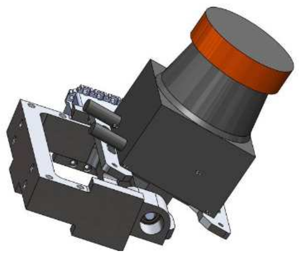


Figure 10. LIDAR mount

C. *Camera*

The onboard camera used is a Unibrain Fire-i Board Pro. The lens used is a fisheye 2.1mm with IR coating. The camera is powered by the 1394 firewire port, which takes a picture at a frequency of 15Hz. The picture is then processed with OpenCV architecture as discussed before. The lane detection algorithm can be best explained diagrammatically:

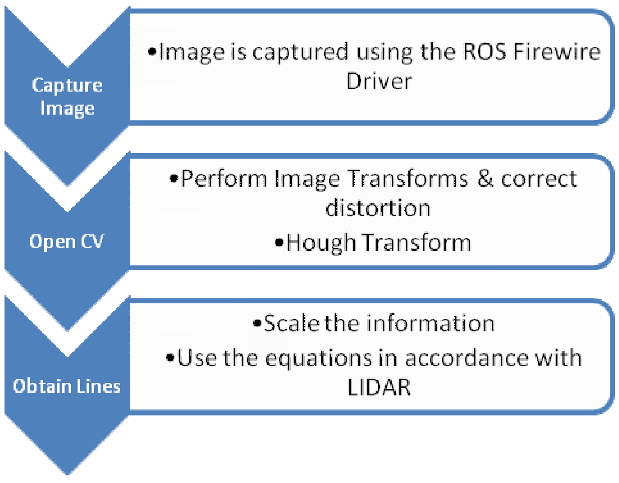

Capture Image
•Image is captured using the ROS Firewire Driver
Open CV
•Perform Image Transforms & correct distortion
•Hough Transform
Obtain Lines
•Scale the information
•Use the equations in accordance with LIDAR

Figure 11. Lane detection algorithm

The images are obtained and processed, to remove any barrel distortion. Processing of the information is done via the well known Hough Transformation. Hough Transform gives us the lines in the (ρ,θ) format, where rho describes the perpendicular distance of a line, and theta is the angle of rho in relation to the pole.

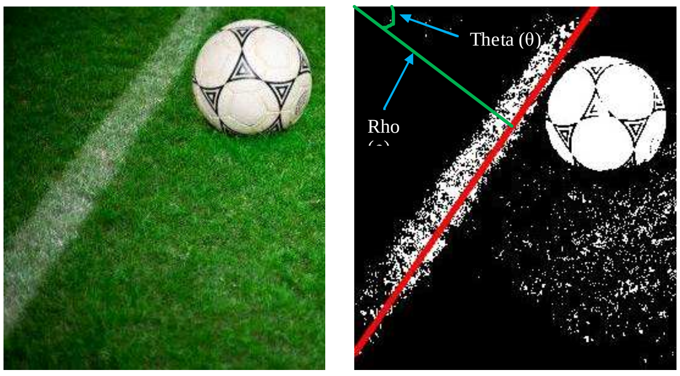


Figure 12. Hough transform example

Figure 12 shows the results of the lane detection algorithm on a sample image. The red line indicates the line detected by the algorithm. The rho and theta values are explained in the diagram. The se values are then passed to the navigation stack to constrain the final path.

D. *IMU/GPS/Gyroscope*

To get the inertial accerelation and the position data, LEO10 uses the Landmark 20 IMU/GPS module. Although the IMU provides orientation and the vehicles angular velocity, the stability of the gyroscope by CruizCore XH1010 was significantly better i.e. 10°/hr compared than to the IMU which was 30°/hr. ROS interfaces with the devices through the USB channel/extracting the data serially. We use an Extended Kalman Filter node to fuse the odometry values from the encoder, orientation data from the CruizCore gyroscope, acceleration and position data from IMU/GPS module

For Pose Interpretation, the node uses 'relative' difference in information of these sensors, and updates the Kalman Filter parameters. Over time, the covariance in the odometry data will increase and grow out of bounds. Hence the node measures difference in this covariance over time (Covariance Interpretation), and uses the information to dynamically adjust the measurements as well as maintain minimal noise to accurately determine robot position and pose at any given point in time.

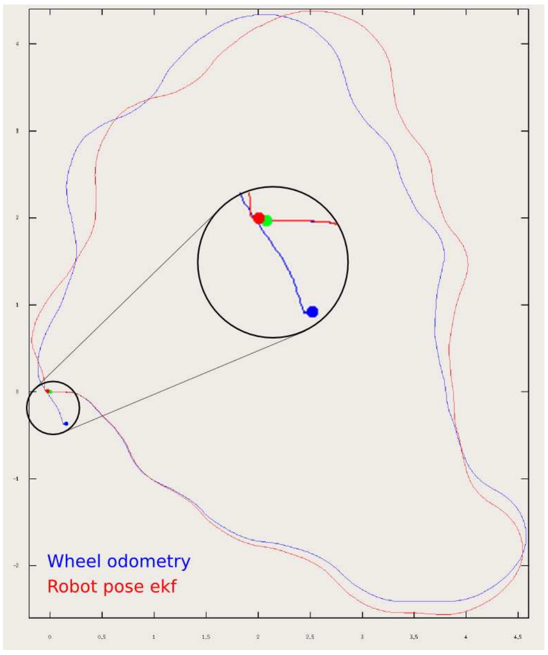


Figure 13. Example of EKF , source: www.ros.org

X. Estimated Performance

Parameters	Estimated Performance	Test Performance
Speed	0.5m/s	0.45m/s
Ramp climbing	Climbs at 0.3m/s	Climbs successfully with varied speed performance
Reaction times of E-stop	0.1s	~0.5s
Battery Life	2h 38mins	~2h
Distance obstacles detected at	3m	~3m
Waypoint accuracy	Tolerance 0.1m	Tolerance 0.25m

XI. Conclusion

The development of LEO10 from scratch was a challenging task for the 3 member undergraduate team. The aim behind LEO10 is to make it more than a successful all terrain robot with navigational capabilities, to make it a stable platform for robotics research and applications in future. Effort has been made to incorporate latest hardware, software and mechanical design and come up with a robot will set high standards in IGVC in future. There are many design features that have been planned to be implemented in the coming future versions of the robot.

XII. References

Krzystof Kozlowski, D. P. (2004). MODELINGANDCONTROLOFA4-WHEELSKID-STEERINGMOBILEROBOT.

Marder-Eppstein, E. (n.d.). ROS Navigation Stack. Retrieved 4 20, 2010, from http://www.ros.org/wiki/navigation

Meeussen, W. (n.d.). ROS: robot_pose_ekf. Retrieved 4 12, 2010, from ROS Wiki: http://www.ros.org/wiki/robot_pose_ekf

XIII. Acknowledgements

We would like to thank DSI, A* Star for funding us for the project and providing us with lab space and mentorship. We are grateful to Mr. Vikas Reddy for guiding us throughout the development and lending us his immense experience while acting as a senior advisor for the group. We would also like to thank Prof. Prahlad Vadakkepat for helping us in liaising with NUS and making the project a success.