The University of Texas at Austin
Robotics and Automation Society
Proudly Presents:

DoloRAS

IGVC 2013

Team Members:
Xihan Bian - Junior, EE
Joshua Bryant - Sophomore, EE
Chris Davis - Freshman, EE
Blake Garza - Junior, EE
Lucas Henderson - Senior, EE
Zichong Li - Junior, EE
Cruz Monrreal - Senior, EE
Robby Nevels - Senior, CS/CE
Manuel Philipose - Sophomore, EE
Andrew Surratt - Sophomore, ME
Sagar Alok Kumar Tewari - Senior, CS
Frank Weng - Senior, CE

I certify that the engineering design present in this vehicle is significant and equivalent to wor that would satisfy the requirements of a senior design or graduate project course.

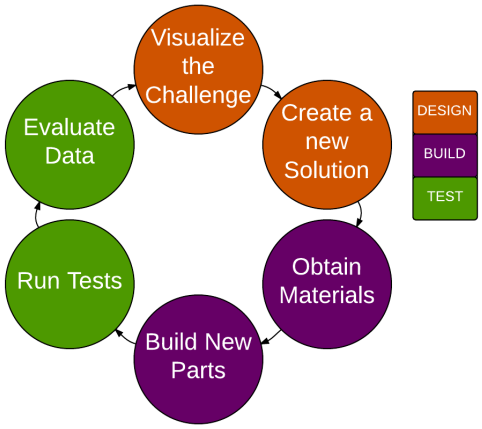
Signed,

_____, Dr. Jonathan Valvano, Faculty Advisor

OVERVIEW

The purpose of the following report is to provide an overview of DoloRAS, The University of Texas at Austin IEEE Robotics and Automation Society's (RAS) submission to the 2013 Intelligent Ground Vehicles Competition. During the past two semesters, we have designed, built, and thoroughly tested our entry for this year's competition. In this report, we describe the systems of DoloRAS, including the mechanical structure of the robot, the electrical and power distribution system, embedded platforms, and software. We will conclude with a list of costs and an analysis of our current performance.

DESIGN PHILOSOPHY AND PROCESS



Our approach to solve the IGVC problem was to break 'autonomous navigation' into smaller testable challenges that could be tackled by smaller teams comprised of experienced seniors and younger members interested in helping with the challenge. For each smaller task we followed an iterative process, because when designing hardware and software it is almost impossible to keep in mind every single use-case and execution path. Hence, we made a conscientious effort to keep our system in isolated layers of abstraction, allowing us to test multiple sensors and choose the ones that give us the best results based on tests.

MECHANICAL SYSTEM AND SENSORS

Base

In deciding what to use as the base of our robot, we wanted to keep the chassis as simple as possible. In the past, we have had difficulty with the robotic base when it was custom built, which resulted in more time being allocated to get the hardware working before the software. Because of this, we decided to find a chassis that would not only be already built, but was also cheap and reliable.

We settled on using a Jet 3 Power wheelchair. This wheelchair was built to carry a 300 lbs person at a speed of five mph. It is sufficiently stable and robust to meet our requirements. The

wheelchair operates on a differential drive train, having two motors on either side. This means that rotations are handled by turning the two motors at different speeds, allowing for a zero point turn radius.

A few mechanical modifications were required to turn the wheelchair into a robotics platform. First, we removed the chair portion and replaced it with a custom-built aluminum frame. Our frame is designed to house our electronics, including the computer case, power supply, sensors, and other components. Our frame also has a tall mast to mount our GPS receiver and camera, which requires a tall viewing angle. It also has a convenient location to put the payload directly over the driving wheels, as well as places to mount peripherals for testing, such as a monitor, mouse, and keyboard.

Sensors

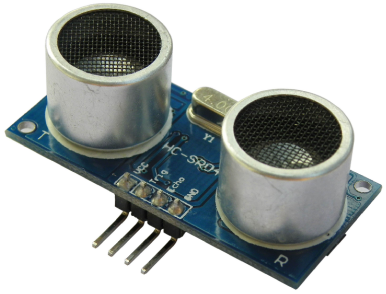
DoloRAS uses several sensors in order to localize and model the environment around it.

The camera we use is a Logitech Pro 9000 with a recording resolution of 1600x1200. In addition to being cheap, it is easy to interface with. However, it produces poor images when the robot shakes or is in bright lighting conditions.

We use a Hokuyo UHG-L8X LIDAR sensor with a detectable range of .2 to .8 meters, scanning range of 270 degrees and .36 degree angular resolution.

In addition, we use a VectorNav-Rugged 200 GPS/IMU. This module allows for 2m GPS accuracy when in motion and 5 meter accuracy at a stand still. The GPS receiver includes 50 channels. This module also has a gyroscope with a measuring range of 2000 degrees per second at a bandwidth of 256Hz, an accelerometer with a range of 8g and a bandwidth of 256Hz, and a magnetometer with a range of 2.5 Gauss and a bandwidth of 200Hz.

Unfortunately, our Hokuyo LIDAR occasionally malfunctions and crashes multiple times on startup. As a backup, we have also arranged an array of sonars for detection of physical obstacles. These are 12 sonar sensors spread evenly over 180 degrees in the front of the robot that emulate the functionality of a laser scan. They are less reliable however, and occasionally interfere with each other due to cross talk.



POWER DISTRIBUTION

The first consideration in our power system is safety for both the machine and its operators. As such, the first step in constructing a power distribution system is a way to shutdown the system in case of accidental actions that can cause harm to anything around it or to the robot itself. This kill switch system consists of a main relay which cuts power from two lead acids to the motor controllers, a large red button which is easily visible and accessible, and a remote relay which can turn the motors on and off from over 100 meters away. All components of the kill switch system have a safety factor of at least 5 in terms of current draw. The output of the main relay goes to the Victor motor controllers, which are full h-bridges capable of supporting 80A of continuous current at 24V, with back-emf protection. Their outputs go directly into the motors and are controlled by an PWM signal.

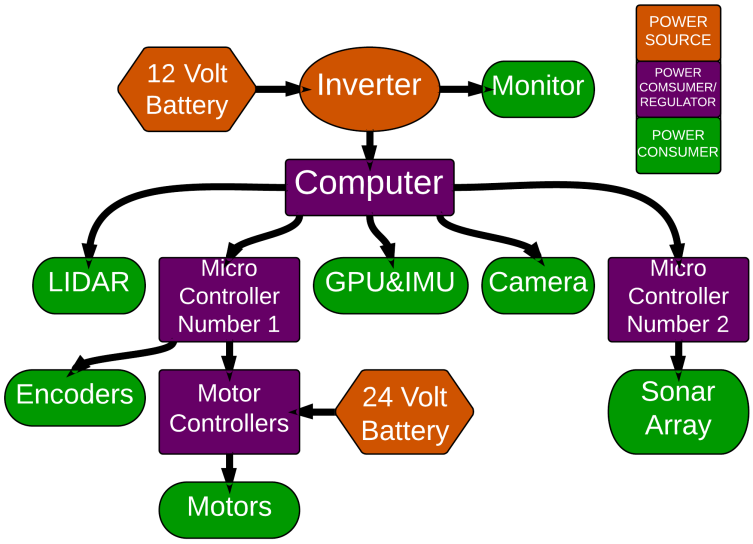


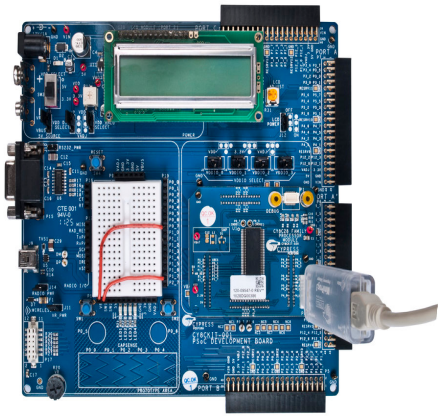
Figure 1: Power distribution system

A separate power system, decoupled from the large current required to power the motors, is powered by another single 12V lead acid battery. This lead acid plugs into an inverter which

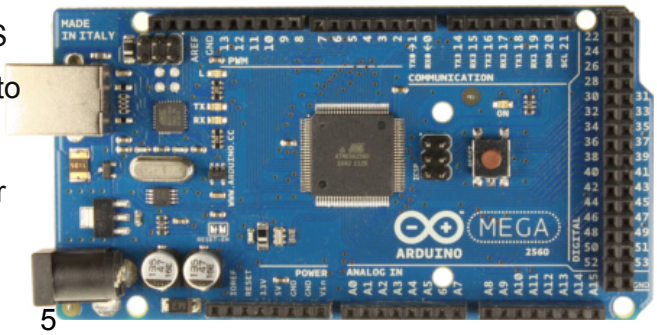
outputs 120V AC power used by the monitor. The computer power supply goes into a power multiplexer along with a connection to the lead acid battery. This allows for “hot-swapping” the computer’s power between the wall and the on-board batteries. The multiplexer connects to a DC to DC converter outputting generic computer power lines, which plug into the motherboard. This power is also split up into a 12V and a 5V Anderson powerpole rail, which can be used to power various peripherals requiring those voltages. The rest of the peripherals are powered by the 5V USB power bus. We use Anderson Power Pole Connectors as a standardized plug for all power connections throughout the robot.

EMBEDDED SYSTEM

Our primary embedded system is the PSoC 5 Development Kit donated by Cypress. The PSoC 5 is a Programmable System on Chip that has an Arm Cortex M3 hard processor and logic blocks that can be configured through an interface provided by Cypress. Our decision to use the PSoC was based on ease of use, since the PSoC not only has a powerful CPU, but is also highly configurable and maintains a high level of flexibility without a steep learning curve.



The current function of the PSoC is to acquire and process the raw data from the encoders and control the motors. It does this while providing an abstraction to control the motors by accepting linear and angular velocity commands from the main computer. The PSoC then attempts to move the robot at the provided velocities using a proportional-integral-derivative (PID) loop which matches the encoder inputs to the motor outputs. While this is happening, the PSoC also abstracts the encoder feedback with angular and linear velocity readings sent back to the main computer.



The second embedded system on DoloRAS is an Arduino Mega whose only purpose is to receive sonar data and pipe it back to the main computer. The PSoC was not used for this because the serial connection from the

computer to the PSoC would reach a point of saturation. If we sent any more data, the serial connection between the PSoC and the computer would crash. We needed an embedded platform that had a lot of IO, but because we were short on time and did not have enough room for another PSoC 5 Evaluation kit, we used an Arduino Mega that we had on hand. The Arduino's function is simply to control each sonar in the array and feed the data back to the main computer.

Both the PSoC and Arduino communicate to the main computer using USB-UART with ASCII characters. The choice of using ASCII over some form of encoded data is for easily human readable communications (at the cost of bandwidth consumption).

SOFTWARE

The software running on DoloRAS is a result of several choices made throughout the course fo the project, ranging from the basic platform and toolchain of development to use of third-party libraries. Familiarity with each tool and programming language was an important part of our decisions, however we did not shy away from learning new APIs that fit our needs more closely or were strongly recommended by experienced developers.

Toolchain

Early on, we chose to use the Robot Operating System (ROS), an open-source, meta-operating system that runs on top of Ubuntu 12.04. ROS provides useful hardware abstractions and primitives for passing messages, allowing for simple interprocess communication. Processes in ROS are encapsulated by nodes which share information using ROS messages. ROS achieves interprocess communication by allowing node to publish and subscribe to messages from topics. ROS also provides us with the ability to create ROS services, which wait for requests from client nodes and respond with messages. Finally, ROS has several debugging tools that help us to monitor system performance and store all messages passed in an execution in the form of 'rosbags', allowing us to collect data from sample runs and then test code without having to physically move our robot for testing.

Third-Party Libraries

Another main advantage of using ROS is the availability of third-party libraries, such as drivers for our camera and laser range finder as well as *rosbridge*, a package that allowed ROS

messages to be accessible over the internet. *rosbridge* and *mjpeg_server* enabled us to set up a debugging platform to visualize the data we were receiving from our sensors remotely with very low latency.

However, we tried to stay away from libraries that did too much “magic,” such as the ROS navigation and mapping stack. Instead, we prefered to develop our own method for navigation because we were able to understand and configure it much easier.

Programming Languages

ROS offers full support for Python, C++ and Lisp in the form of client libraries. We primarily used C++ and Python for the main system, along with extensive use of JavaScript to prototype methods of obstacle avoidance and localization. *rosbridge* provided us with an API to publish and subscribe to ROS topics remotely, giving us the flexibility to use JavaScript running on a separate client machine if case we ran into problems with high loads. We chose to use Open Source Computer Vision (OpenCV) as it provided support for C++ and Python and was well integrated in ROS, with features to avoid creating redundant copies of image messages, and compressing in-transit messages. OpenCV is an open-source project maintained by Willow Garage that has been used successfully in robotics competitions in the past, most notably on Stanely, the winning entry of the DARPA Grand Challenge Vehicle. In addition, OpenCV has a thorough GPU port using CUDA, which we used to offload our vision processing algorithms and free up the CPU.

Drivers

While we used the drivers for our webcam and Hokuyo laser range finder found in third-party ROS packages, we also spent a substantial amount of time writing drivers for other sensors we used and tested with. We wrote drivers for the Pololu UM6 IMU, U-blox 6 GPS, and VN-200 IMU which provide GPS and IMU data. We ultimately chose to use the VN-200 IMU from Vector-Nav due to its precision and accuracy. Apart from these sensor drivers we also wrote code to interface and drive the PSoC and Arduino microcontrollers to obtain data from encoders and sonar, and to provide an abstraction for low level control of the motors.

Vision Pipeline

We chose to use OpenCV in Python to develop quick prototypes to tackle the various challenges

with vision, i.e. obstacle detection, lane detection, homography perspective transform and polar transforms to estimate distances using images. Because we are using an Atom processor, we had to be very cautious of our CPU usage and when possible used the GPU to relieve load. With this in mind, we used *cv_bridge*, a ROS package to compress images and minimize copying during interprocess communication. Our vision pipeline consists of separate nodes for detection of obstacles and lanes, since the processing steps for each varied enough to justify the creation of separate nodes.

Thresholding for Obstacle Detection

The node for obstacle detection throttles the number of frames outputted by the camera, processing only the latest image available. This was a design choice to allow us to easily test how far we can cut back on frequency of publishing processed image messages without significantly affecting out higher level decision making process.

For colored obstacle detection we first blur the image we receive from the camera, and translated the blurred Red-Green-Blue (RGB) image into Hue-Saturation-Value (HSV) color space and split the channels in this image to threshold their values individually. The HSV color space provides us with a natural visualization and allows us to succinctly and accurately describe what it mean for an obstacle to be a certain color by using static thresholds.

White Line Detection Using Hough Transform

We chose a two step process to detect white lanes. Our overarching idea was to first identify pixels that could potentially be a part of a white line and produce a binary image with this pixels marked as white and the rest of the image, black. Next we used OpenCV's Hough transform to mark the best fit line on the binary image produced by the previous step. We created a UI in order to vary these thresholds with ease and adapt to lighting conditions with ease.

Homography and Polar Transform

Once a binary image is generated containing lanes and obstacles to be avoided, it must be converted into a format that is usable to the high-level decision making software. We decided to convert all environment-sensing data into a scan format. This requires a homography perspective transform in order to transform the image to a bird's-eye view. We use a chessboard as seen in figure 2 below to automatically determine the homography transform

using OpenCV function calls. Notice how the chessboard in the transformed image appears to be made from square tiles whereas the original image contain rhombi.

Figure 2: Perspective/homography transform

Then, the binary image is converted into polar space using OpenCV's log-polar transform function, shown in the right panel of figure 3 below. In this new image, rows represent angles and columns represent distances. The image is scanned to produce a series of 20 (angle, distance) pairs spanning 180 degrees in front the robot. Use of this scan is described the Obstacle Avoidance section below.

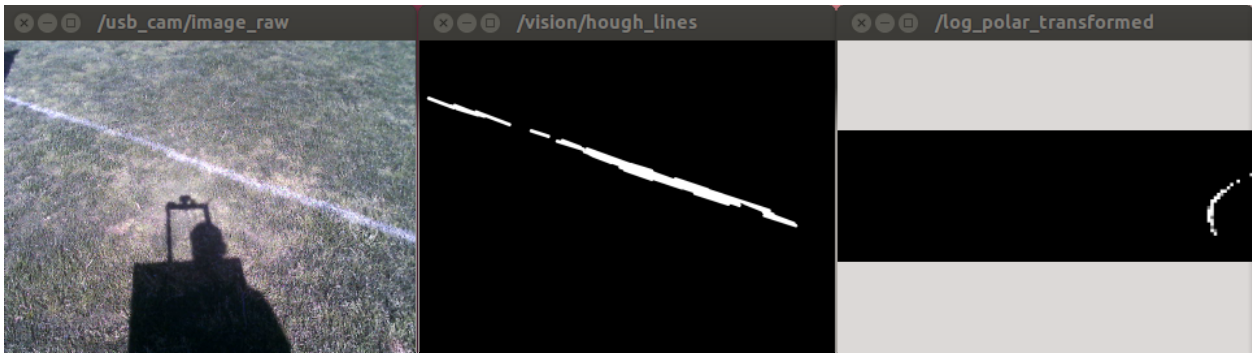


Figure 3: middle: detecting lanes using hough lines, right: log-polar transform on binary image

Localization using an Extended Kalman Filter

DoloRAS uses an Extended Kalman Filter in order to determine its location in the world. A Kalman filter is a linear estimator that uses updates from various sensors and confidence levels in each sensor to produce a state estimation vector. It uses a linear transition of robot's state

from one prediction to the next, while at the same time maintaining a matrix of variances, or confidences, corresponding to each element in the state vector. This state state estimation process is robust because it can tolerate errors or even failures in a set of sensors and still provide a reasonable state and error estimation. An Extended Kalman Filter (EKF) approximates a nonlinear state transition function, which is more suited for a differential drive robot like DoloRAS. For more information on Kalman filters and EKFs, see [1].

Rather than using an existing implementation, we decided to write our own EKF to have flexibility on how state estimates are computed and updates are used. Our EKF estimates a state vector that consists of position (in meters), heading, linear and angular velocity, linear acceleration, roll, and pitch. We used the VN 200's GPS sensor for global position updates (using x/y meter updates computed from GPS coordinates using a local reference point), the VN 200's accelerometer and magnetometer for roll/pitch/yaw updates, and the motor's encoders for angular and linear velocity updates. We calculated sensor uncertainties from the GPS and IMU by analyzing data while DoloRAS was at rest, and estimated the encoders' uncertainty from observing how much drift occurred when calculating the robot's position directly from the encoder values.

Once the EKF was completed, we found that there was very little drift while using only the encoders (approximately 1 meter for every 20 meters travelled), unless slippage occured on grass, which could be somewhat compensated for when using directional updates from the IMU. With everything combined, we are able to achieve a smooth path estimation which has has an overall approximate standard deviation of error under 2 meters. This is significantly smoother than using GPS updates exclusively, as seen in the diagrams below.

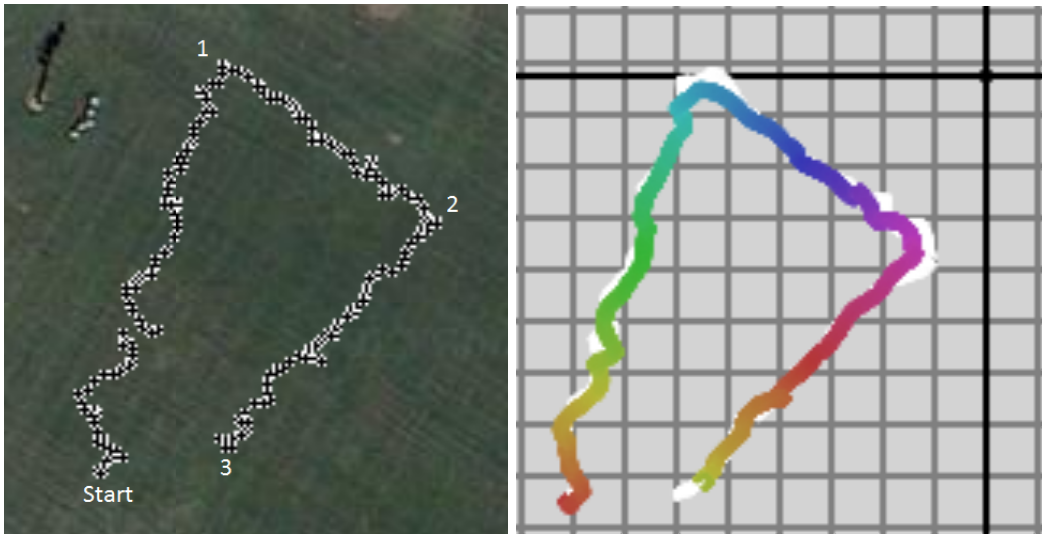


Figure 4: The EKF is tested on an outdoor soccer field. Right: GPS updates; left: EKF estimate using combined encoders, IMU, and GPS updates. The boxes on the right are 4 by 4 meters.

Obstacle Avoidance and Decision-Making

In order to use data provided by sensor in the decision-making process, we first convert the data provided by each sensor into a scan format, making it simple to combine scans and abstract details of the sensors from the decision-making code. For this, we used the Hokuyo UHG LIDAR scanner, a custom-built array of sonars, and a webcam. Creating scan data from the Hokuyo and sonar array is trivial, but creating a scan from the camera data requires the vision pipeline described in the vision section above. An scan can easily be generated once the log-polar transformed binary image is published. An example of this can be seen in figure 5 below. This is a continuation of figure 3 in the vision pipeline section.

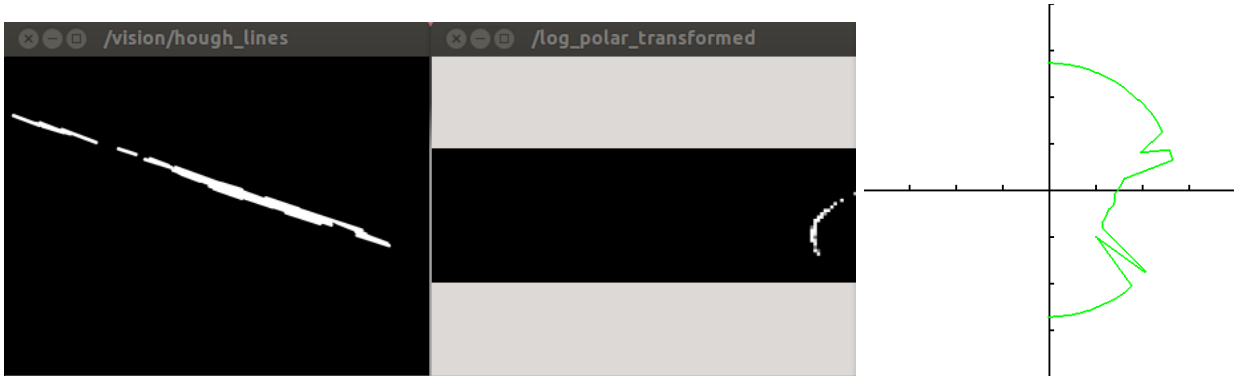


Figure 5: showing an image scan derived from the same image as Figure 3 above

Once a combined scan is created, we use a modified version of the Dynamic Window (DW) technique described in [2] which takes in the current goal point and the state estimation output from the EKF. Although this is a purely reactive technique, we believe that it will be sufficient for navigation at the IGVC. During the competition, we will likely add extra waypoints inbetween the given ones to allow the robot to move smoothly through the field.

Our DW approach samples angular and linear velocities in a window around the current estimate of the robot's angular and linear velocities (hence *dynamic* in the name). Each sample (linear v, angular w) is weighted using four criteria, and the best is chosen to be sent to the PSoC driver, and on to the motors. Figure 6 shows an example of the angular and linear velocity sample trajectory space.

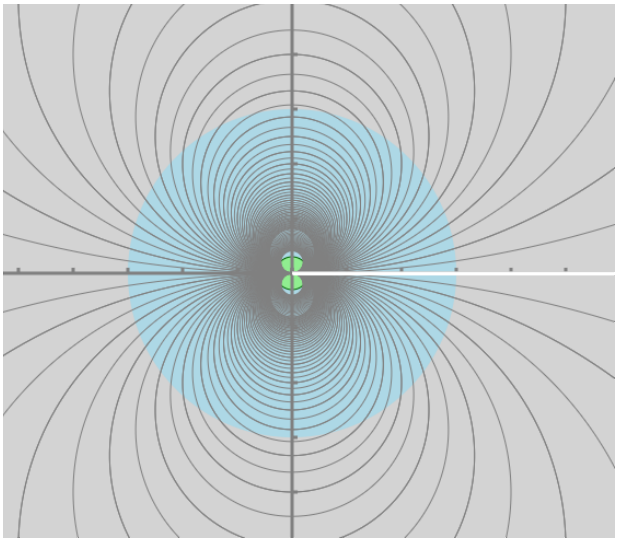


Figure 6: An example of the angular and linear trajectories calculated with the robot at rest, facing towards the positive x-axis. Each axis show ticks every meter.

We used the following criteria to weigh samples:

- Maximizing linear velocity
- Minimizing the difference between the direction to the goal and the heading in the trajectory after a time step
- Maximizing clearance, the distance that the robot can follow a trajectory before hitting an obstacle. See the figure 7 below.
- Minimizing goal alignment distance, the minimum distance to the goal from any point on

the trajectory. See the diagram below. This weight was added from the original DW technique in [2] because it helps prevent the robot from swerving back and forth when heading towards a goal.

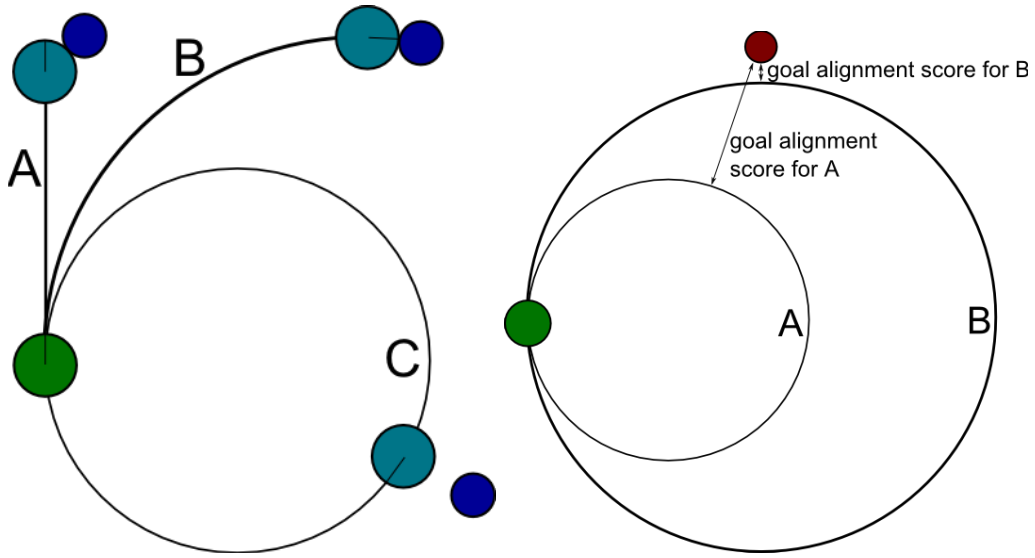


Figure 7: Right: A, B, and C are the clearance distances calculated for each trajectory. Left: the goal alignment values are shown for each arc A and B.

We originally implemented the DW code in Python, but decided to switch to Javascript in order to speed up prototyping and visual inspection of the process. We use *rosbridge* to transfer data and commands to and from a browser. This also allows us to easily connect a much faster computer to our current one and offload the decision making computation (the extra computer only needs a browser and an ethernet connection to the main computer in order to run the program). We have currently measured the upper bound of the data latency between the main and extra computer to be approximately 150 ms.

PERFORMANCE

We have observed our robot travelling to multiple GPS waypoints while avoiding obstacles and using all sensors to localize. The robot performs poorly in cluttered environments, occasionally ignoring small obstacles. This is due to the noise in scans that are produced from the vision processing nodes as well as the latency in the system. After migrating the software to a faster i5 processor, we expect that this problem will lessen.

We have benchmarked performance gains when moving our image processing algorithms from the CPU to the GPU. See the results of a case study of doing Canny edge detection in figure 8.

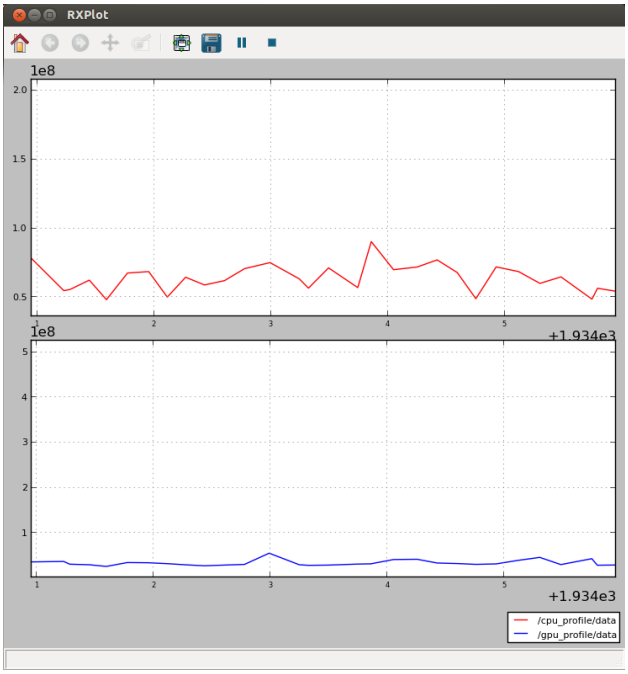


Figure 8: Graphs of CPU vs. GPU usage while running Canny Edge Detection

COST

Item	Quantity	MSRP (USD)	Cost (USD)
Jet 3 Power Wheelchair	1	$1500	$40
Hokuyo UHG-08LX LIDAR	1	$3950	$0
U1 12V 35Ah Lead-Acid Battery	2	$160	$160
1” Aluminum Angle Brackets, ⅛” thick	10	$80	$80
ZOTAC D2550-ITX WiFi Supreme	1	$163	$0
Cypress CY8CKIT-050 Programmable SoC	1	$100	$0
Intel Atom Processor, 1.86GHz	1	$95	$0
M4-ATX DC-DC 250W PSU	1	$90	$0
Logitech QuickCam Pro 9000	1	$90	$0
Ublox AEK-4H GPS Evaluation Kit	1	$200	$0
Anderson Power Pole Connectors	50	$60	$60
Misc. Hardware (Nuts and Bolts)	NA	$100	$100
Quadro 6000	1	$3,999	$0
Hero3 Black Edition	1	$399	$0
VN 200 Rugged	1	$3200	$0
StarTech PCIe HD Video Capture Card	1	$120	$120
2 Channel Remote RF Relay	1	$30	$30
IFI VEX Pro Victor 885	2	$400	$400
HC-SR04 Sonar	12	$36	$36
Total:		$14,186	$703

REFERENCES

[1] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. Cambridge, MA: MIT, 2005. Print.

[2] Fox, D.; Burgard, W.; Thrun, S., "The dynamic window approach to collision avoidance," *Robotics & Automation Magazine, IEEE* , vol.4, no.1, pp.23,33, Mar 1997