Adversarial Strategic Games and Robotic Design

Submitted By:

Griffin Tabor: Robotics Engineering

Jonathan Berry: Robotics Engineering

Evan Gilgenbach: Robotics Engineering and Computer Science

Advised By:

Kenneth Stafford: Robotics Engineering and Mechanical Engineering

Carlo Pinciroli: Robotics Engineering and Computer Science

Table of Contents

1. Introduction

Great strides have been made in improving the decision making capabilities of robots in unpredictable situations, but relatively little attention has been paid to the ability of robots to reason about—and compete with—human and non-human opponents. Such systems must collect and process data on the changing environment, anticipate actions of opponents, and navigate in often cluttered environments. One example of adversarial strategy in robotic design is found in competition robotics, which frequently feature the complexities of multiple actors and difficult maneuvering requirements. However, many of these competitions allow for human operation of robots for portions of matches, in an attempt to limit the effects of challenges in robotic localization and strategic planning. There is currently a great deal of potential for robot-only solutions to these challenges.

In this report, we present prototypes and approaches for four different high-level tasks applicable to small, fully autonomous robotics platforms competing in a college-level extracurricular robotics competition. The goals of the project are real-time motion planning and maneuvering, long-term localization, object detection and tracking in a cluttered environment, and adversarial strategic planning. The MQP team worked in collaboration with the WPI Vex team, with the aim to make the software capable of running on a range of possible robotic components, encouraging similar methods to be used on future robots.

2. Background

2.1 - Literature Review / RoboCup

The closest existing project to ours in scope is the RoboCup competition. This annual competition attempts to improve the state of AI and robotics research through fully autonomous soccer playing robots. Competing robots play a nearly standard game of soccer in an adversarial environment, collaborating with other robots programmed by the same team. Our specific challenge is most similar to the Medium Size league, which allows customized robots, and does not have humanoid competitors. The robots entered in the RoboCup competition are required to do the same object detection, path planning, and adversarial strategic planning as the autonomous Vex project.

Figure 1: A typical Medium Size division RoboCup competition. (RoboCup 2017)

2.2 - ROS (Robot Operating System)

ROS is a collection of tools to create modular robotic control systems. It is a language independent architecture which handles communication between nodes that each handle a specific task, making code reuse simpler. ROS nodes also provide low-level device abstraction, with nodes handling hardware interfacing collecting and sending relevant data to processing nodes. ROS features a large number of existing packages intended for object detection and robot navigation, such as the teb (time elastic band) navigation planner, OpenCV, and Stereo Visual Odometry (SVO 2.0). Here the ROS Navigation stack, an example of a collection of nodes communicating to perform a specific task. This modular design means any robot design can run all the same navigation code and low level abstraction is left to robot specific implementations.

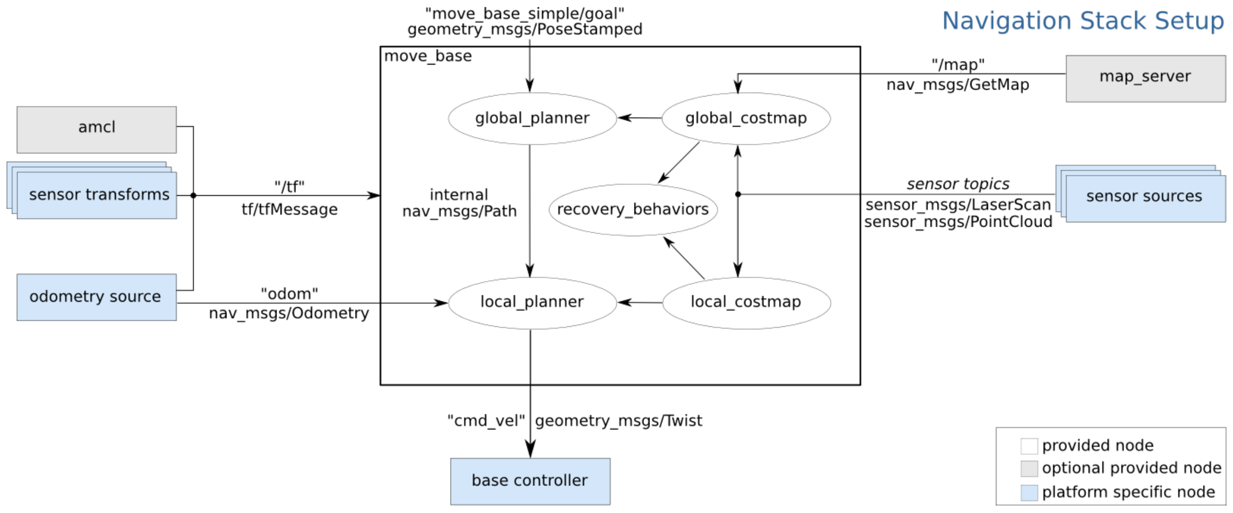


Figure 2: A diagram showing ROS nodes in the Navigation Stack. (ROS 2017)

2.3 - LIDAR Sensors

LIDARs[1] are a type of sensors that project light and measure the return light that bounces back off of a surface to determine an object's distance away. Various LIDARs use different techniques such as time of flight measurements and intensity of returned signals to calculate distance to object. Most traditional LIDARs spin so a single emitter/ receiver can be used for measuring multiple directions. The angle of the emitter and the distance to the object is combined to produce a point relative to the sensor. This simple class of LIDAR is known as 2D lidar as it only measures points in a plane orthogonal to the axis of rotation. Many applications however require a 3D point cloud of the environment so there are two main techniques: the first is to mount multiple spinning emitters/receivers as to measure multiple planes, the second is to take the 2D lidar and mount it on another axis of rotation so it always measures a different plane. The multiple emitters technique is extremely expensive because of the duplicated components. The other technique of course sacrifices its update rate because the sensor has to alternate which plane it is scanning.

2.4 - Fisheye Camera

A fisheye lense is a type of lense that bends light from a full 360 degree view into a standard camera view. Traditionally these cameras have 360 degree by 180 degree by 180 degree field of view and can see a full hemisphere. Many robotics systems use fisheye cameras to get a large field of view with a lower number of cameras. A sample image from a 1080p fisheye camera pointing down can be seen on the next page.

[1]LIDAR as an acronym is not well defined. We have found references to it in the literature for standing for ‘Light Imaging Detection And Ranging’, ‘LIght Detection and Ranging’, ‘Light Illuminated Detection And Ranging’, ‘Laser Illuminated Detection And Ranging’, and ‘Light RADAR’ where DAR was used as part of a word, rather as an acronym.

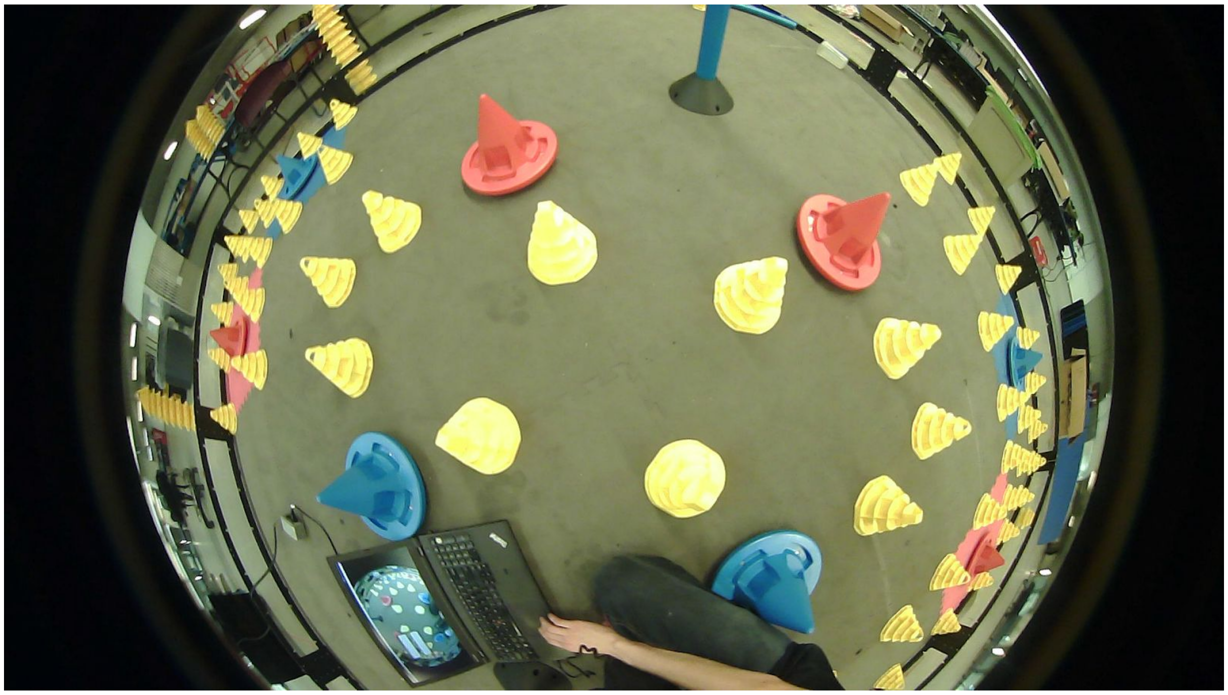


Figure 3: A view of the project's environment using a fisheye camera.

2.5 - Q-Learning

Q-Learning is an approach to autonomous decision making that records the *expected value* of each state possible in a discrete state graph. While the Vex game is not a discrete state graph, we believe it can be modeled as such for the purposes of strategic decision, as is common in AI application. Q-Learning is an approach which takes a large dataset of game playthroughs and builds up a mapping from a (state, action) tuple to the expected value (the probability of winning the game if you take the given action in the state).

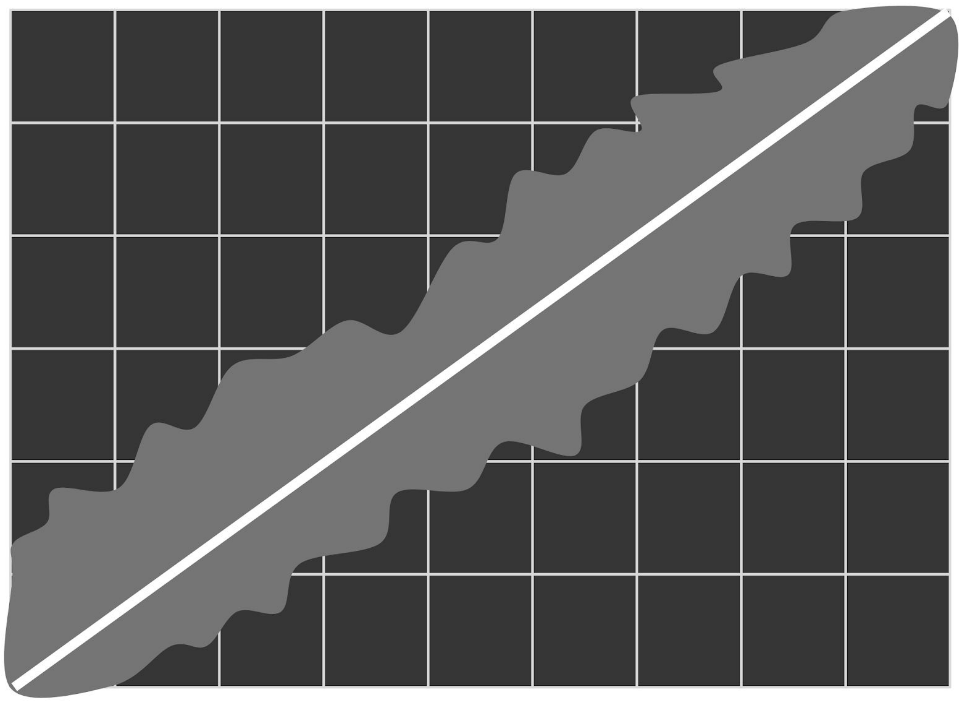


Figure 4: An illustration of the random walk (light grey), guided by the reflex agent (white)

To generate a large dataset of game playthroughs, a common method is to create a reflex agent, and used an epsilon random-walk method to take a "random" action with some small probability. So, for example, at each state the computer would simulate the reflex agent with probability *0.8*, but simulate a random action instead with probability *0.2*. This allows you to build up a dataset covering a large part of the useful state space without spending an inordinate amount of time covering the entire state space.

2.6 - Tensorflow

Tensorflow is a neural network training framework optimized for deploying onto a GPU. Neural networks work similarly to how a human brain functions. Tiny computation units called neurons sum up a set of input neurons values multiplied by their weights. Stacks of these neurons can be tuned to model more complex equations. Convolutional Neural networks have consistently shown promise dealing with images. Networks take in images or pieces of images and a label of what is in the image and learns to label new images. The hard part of neural networks is designing the structure of neurons to train. A structure of 1000 layers of 1 neuron will certainly perform worse than a structure of 10 layers of 10 neurons each even though it only has 100 neurons. TensorFlow released a set of structures that they found to be ideal for object detection. The structures were trained on millions of labeled images and can be used to initialize a network for any other image recognition problem. Below is an example of the TensorFlow object detection API running on an image at the beach after being trained on the COCO dataset.

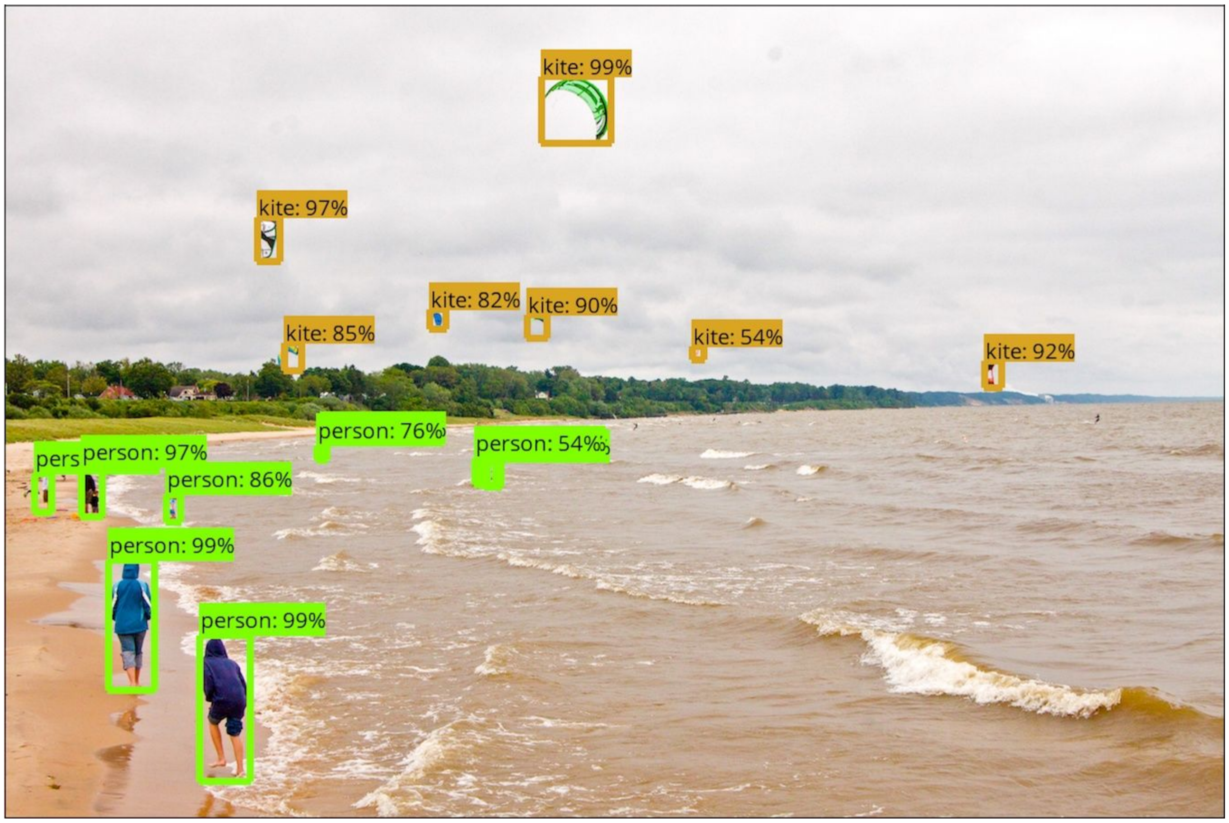


Figure 5: An example of Tensorflow-created labels superimposed on the processed image. (Huang et al. 2017)

2.7 - SLAM & Google Cartographer

In robotics systems mapping and localization interact in a unique fashion. Given a perfect understanding of what the world looks like and a way of perceiving the world, state of the art systems can determine every possible robot position that would get the same sensor data. Given a perfect position in the world and a way of perceiving the world, state of the art systems can build a map of the world as it moves around. The idea of SLAM, simultaneous localization and mapping, is that a partial map and a guess of current position can be used together to build a more accurate estimate of both. These systems log the positions of various things as the robot moves around and use the estimated robot position to plot the map in the world. The map it builds is then used with the the position guess to refine the position. Newer techniques even attempt to readjust all old generated maps with corrected positions of the robot after it finds a location it has already seen. Google recently released Cartographer a general use SLAM framework that focuses on combining wheel odometry and 2D lidar scanners. It also has a few unique features, like the ability to lock the map into a set configuration and only localize within it and the ability to load an old map from memory. Image below is an example of Google Cartographer mapping a museum running off 2 lidars mounted on someone's backpack as they walk around.

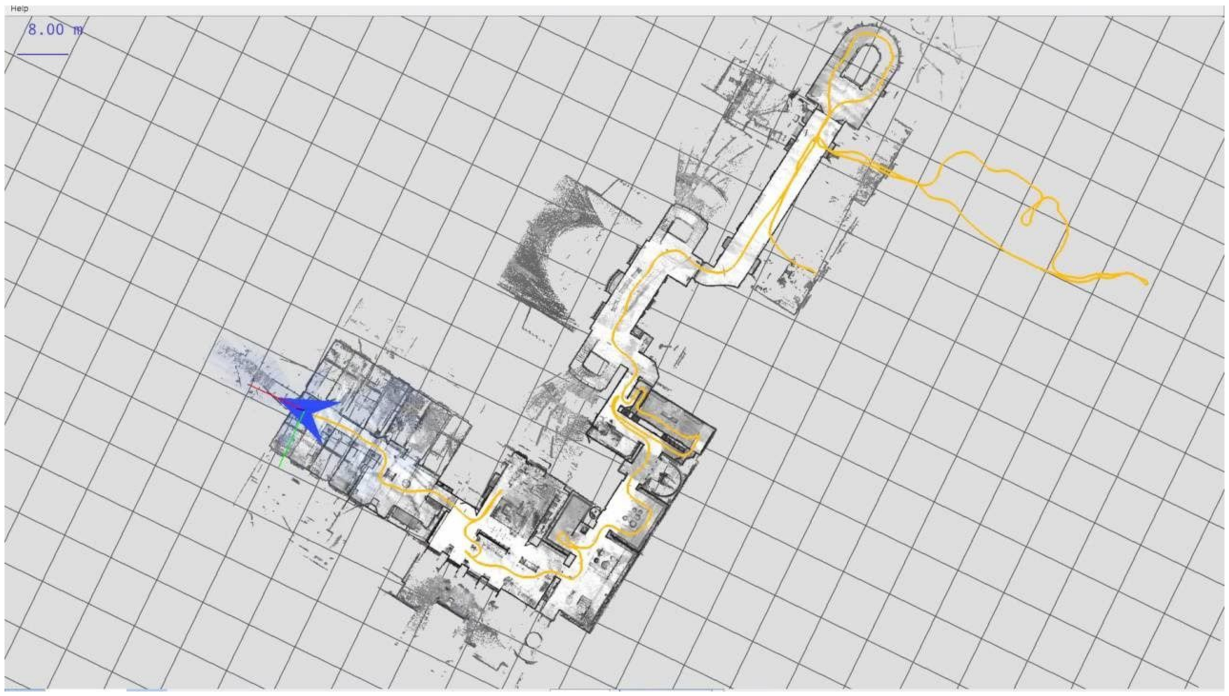


Figure 6: A model of a building produced using LIDAR data and Google Cartographer. (Google Cartographer 2017)

2.8 - Vex U Robotics Competition & WPI Robotics Club

Vex Robotics is a educational robotics equipment company, which sells kits of equipment intended for classroom use in middle and high schools. Vex Robotics also organizes the annual Vex Robotics Competition, which has divisions for middle school, high school, and collegiate participants. For the collegiate division, Vex U, games are played on a 12’x12’ field, with two opposing teams competing over the same set of scoring objects. Each match features a 45 second autonomous period, followed by a 75 second teleoperation period, for a total match length of two minutes. The 2017-2018 Vex U competition is named “In the Zone”, and features eighty 7 inch tall yellow plastic cones, which competitors must place on their team’s goals. The goals are 10 inches tall, and are shaped to allow the cones to nest on top. The amount of resource contention, combined with the large number of objects on the field at any given time, make path planning, object detection and real-time strategy a significant challenge for an autonomous competitor.

2.9 - Previous WPI Projects

For the 2016 Vex U competition the WPI Robotics Club, in an effort to bring real autonomy to a Vex competition, developed the first ever fully autonomous competition Vex robot. The proof of concept was able to play the entire match for 2 minutes, which had been the goal of the project. One of our MQP members lead the team developing the robot. The Vex game that year had a wall across the middle of the field, between the competing robots. This completely negated direct interaction between robots, and vastly reduced the benefit of adversarial planning. To simplify planning, the robot disregarded all data collected for objects on the opponent's side of the wall. The strategy planned only a single object in the future, and was very greedy in its objectives, grabbing and scoring the closest to the robot instead of going to a slightly farther grouping which could potentially be more strategic. All localization was done using an Extended Kalman Filter merging together wheel encoders and an IMU. Though this method offered decent accuracy for that application, the localization was still entirely done using dead reckoning, by simply summing the past motions to estimate position, and didn’t use the field surroundings at all. The robot was subject to substantial drift if it ever ran into anything or moved in an unmeasurable way. All navigation and path planning was done on the small embedded board and limited to simplistic paths produced using trigonometric distance calculations. Pathing around obstacles was completely hardcoded and so conservative almost ⅔ of our side of the field was considered to be unsafe to enter without a complex maneuver. For object detection, the robot assumed every consistent grouping of lidar points was a scoring object, which heavily relied on the rules of the 2016 competition keeping the opponent on the other side of the wall. The detection system did try to distinguish between small objects and large objects, but used exclusively the radius of the cluster of lidar points, and had no way of differentiating several small objects from a single large one. Through all of these shortcomings the robot was successfully able to play the game for the length of a match, but significant work was needed to improve the system to perform as well as even an inexperienced human.

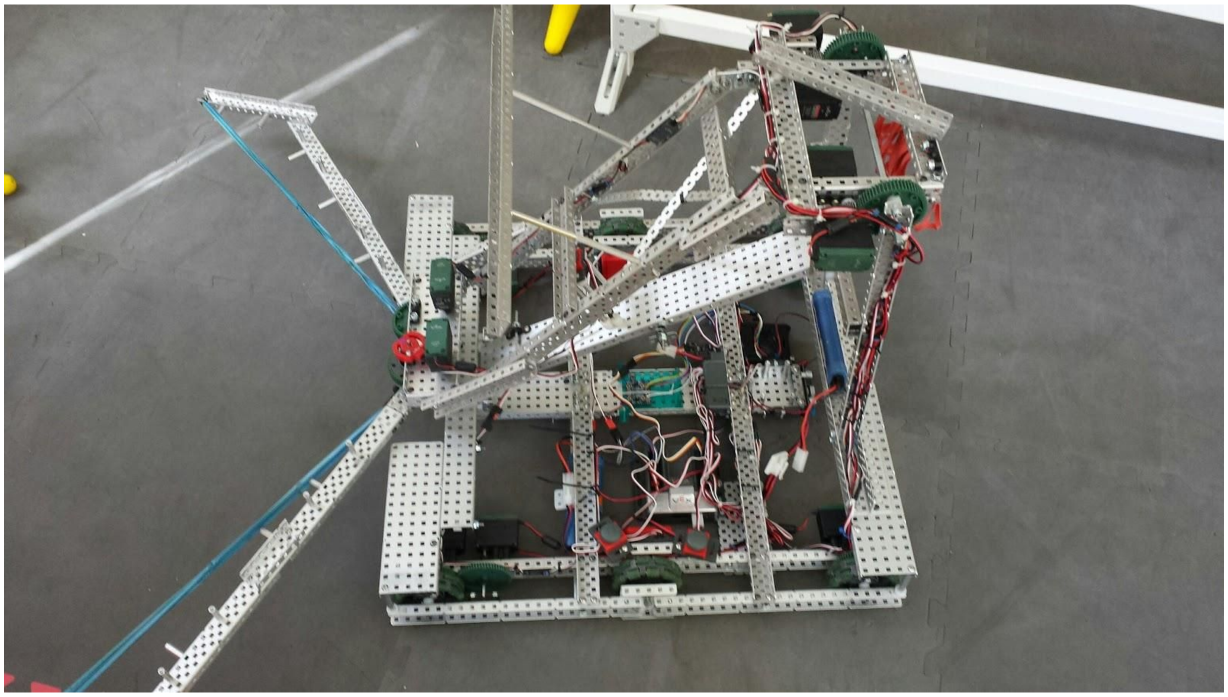


Figure 7: The WPI fully autonomous robot for the 2016 Vex Competition.

3. Project Goals

3.1 - Statement of Work

Project Goal	Quantity	Quantified Deliverable
Navigate a cluttered environment	Speed	Within 20% of the time-optimal path as calculated by the TEB local planner.
	Accuracy	The robot's steady state error will be no more than half the allowable base front tip error (defined below)
Long-Term Localization	Accuracy over time	After 15 minutes, a maximum displacement of 3 inches radius measured from the front center edge of the robot.
Object Detection	Speed & Accuracy	After driving the full length of the field, the system should have measured at least two data points for 80% of field objects and no false positives for objects with at least two false data points
Strategic Planning	Adaptability	While executing a plan, the robot should be able to change its plan and adapt to a missing scoring object in under 3 seconds
	Throughput	The robot should be able to recalculate the entire strategy within 15 seconds

3.1.1 - Navigation

We expect our steady state error to be less than 3” due to the physical design of the robot and the design of the goal objects. Each cone is 6” in diameter, and the robot manipulator can be modeled as being at the center of the front edge of the robot. Therefore, if the robot is more than 3” in radius from the cone, it will be impossible for a manipulator to even make approximate contact. The manipulator compliance is expected to be lenient enough to pick up the cone if it enters within this radius of the front edge center. This also specifies the allowed rotation: since the pose of the robot is defined as a rotation about the center, the maximum allowable rotation (assuming no lateral error) can be derived to be 14 degrees in either direction.

3.1.2 - Localization

Our localization goals are founded on the same principles as the navigation steady state error goals. Given that the object manipulator must be accurate enough to pick up the smallest goal object, we can never be allowed to drift farther than a three inch loci from the expected front edge center position. This is measured as drift over time rather than just maximum allowable error to better model the possible inaccuracies in the system, and ensure that in the 2 minutes of competition we will always be localized.

3.1.3 - Object Detection

The project goals for object detection come directly from the perception requirements of the system. To be able to construct a strategic plan and pick up cones, we first need to create a map of the entire field including scoring objects. This requires precise sensing over time. Predicating the measurement on traversing the entire field is due to the nature of obstructed objects and the necessity of fusing multiple data points over time into a coherent world-state. That is, requiring at least two data points for each object ensures the possibility of *associating* multiple data points into one “object”. The requirement for a lack of multiple objects for false positives is also a combined goal of the perception system (the specificity of the initial data-point detection) and the processing system (the specificity of its ability to discard multiple “false” data points).

3.1.4 - Strategic Planning

The goal for speed of adaptability is to react to small changes in the plan while a step is being executed. Within 3 seconds of an opponent picking up a cone we are planning to get in the future we should have found the cheapest replacement cone to pick up instead. In a match we estimate it takes 3 seconds for a robot to drive and pick up a cone so we expect in the worst case cones disappear around our robot once every 3 seconds.

The throughput goal for the robot is derived from the need to responsibility to high-level strategic decision made by the opposing team. Over the course of the match, teams are expected to score all 4 movable goals, and many of the scoring objects. Since the number of goals is very limited, each goal scoring can be thought of as a major milestone in the strategic development of the game. Given that there are 4 goals in a 2 minute match, each goal should take on average 30 seconds to score. That means we should be able to re-plan twice during each “goal scoring period”—once for the initial plan and once to react to any changes before you finish scoring the goal. This leads us to the expectation of 15 seconds for a full creation of a strategic plan.

3. Design & Results

3.1 - Localization

Our final system design for the localization subsystem fused wheel encoder odometry with LIDAR scan data. This was done by using the "scan matches" from Google Cartographer to update the odometry base offset. The odometry estimated pose was then fed back into Google Cartographer to use as a prediction for the laser scan matching. The configuration for accomplishing this can be found in our project repo, with the majority residing in the file `turtlebot_urg_lidar_2d_localization.lua`.

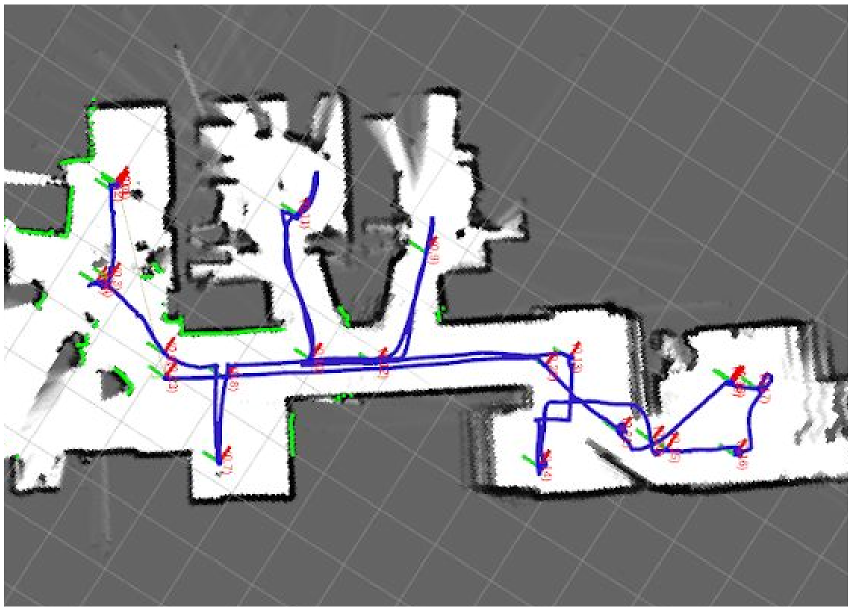


Figure 8: The prototype localization system mapping a dorm room, before odometry

The performance of this localization system was extremely successful for us—over extended, 15 and 20 minute sessions we did not see any appreciable buildup of error over time. This is attributable to the design of the SLAM system, which converges to zero over time as confidence increases.

One problem we encountered was that our use of wheel odometry as a "fast update" system was somewhat ill-suited to the cluttered Vex field. While navigating, the robot would often incidentally run over the corner of an obstacle or bump into a static wall, which would cause localization errors for 3-5 seconds while Cartographer brought the offset back into alignment. While our original design had incorporated the Semi-direct Visual Odometry library (Forster et al. 2017), bugs in the implementation as provided prevented us from using it in our final system. We believe that the successful integration of visual odometry would provide a much more reliable source of short-term, “fast update” position estimation then wheel encoders for future designs.

3.2 - Navigation

Our final navigation system uses the `teb_local_planner` software package (TEB, 2017), combined with the stock ROS `global_planner` package. We implemented a custom node in python (the `executor`), that translated high-level goals (“pick up goal 1”, “score goal”) into concrete poses that were passed to the navigation stack. For populating the navigation system we used another node that simultaneously passed polygonal obstacles to TEB and costmap obstructions to the global planner. These obstacles were static, because the navigation system was developed and tested independently of the object detection system.

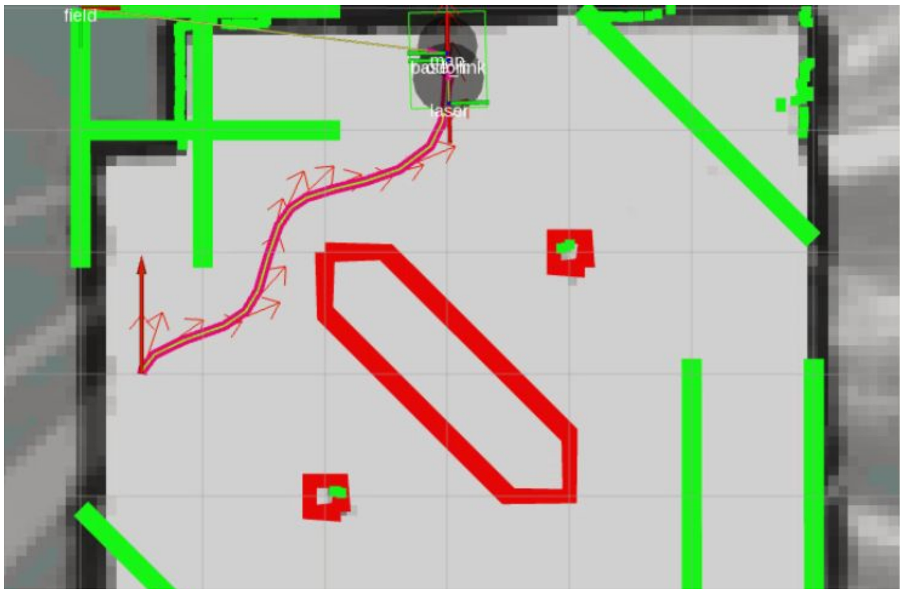


Figure 9: TEB-planned robot poses with overlaid obstacles

One thing that we discovered during our implementation were the limitations of TEB as local plan optimizer. Because it works in a fashion similar to hill-climbing, without a global planner it can occasionally get “stuck” in local cost minima, and be unable to find larger optimizations. This is not a common behavior but it does happen when larger paths (8-12 feet) are combined with large obstacles. We mitigated this issue by improving the global path planner and adding obstacles to the global costmap.

The performance of the navigation system with respect to the fixed-plan navigation test was accurate enough for our purposes—we consistently picked up and scored goals without steady state error. The transient error of the navigation system (when navigating from setpoint to setpoint) was accurate enough to navigate between the crowded field corridors, but occasionally caused our wheels to run over cones, which would itself cause odometry errors. While this affected our consistency slightly, it didn’t manifestly damage our design goals, and we felt it was a desirable tradeoff for the speed required. Future mechanical design could do a better job at shielding the wheels from game objects, like many Vex robot designs do.

While we did not manage to solve the full extent of the field-traversal problem mentioned for larger paths (even with the global planner), we’re confident enough in the accuracy of the other parts of our navigation that we think this is a viable approach. See the included video.

3.3 - Object Detection

Our final object detection system uses a customized version of the tensorflow `object_detection` package, which provides base code for implementing a routine to train object detectors. For collecting the data, we used the Vatic software project (Vondrick 2012) to provide a robust video-tagging solution that allowed the entire team to work on the labeling problem simultaneously. We ended up with 4,848 labeled images and around 70,000 labels. Of the four-thousand labeled images, we were unfortunately only able to use around 1,000 for our final detection system due to time constraints.

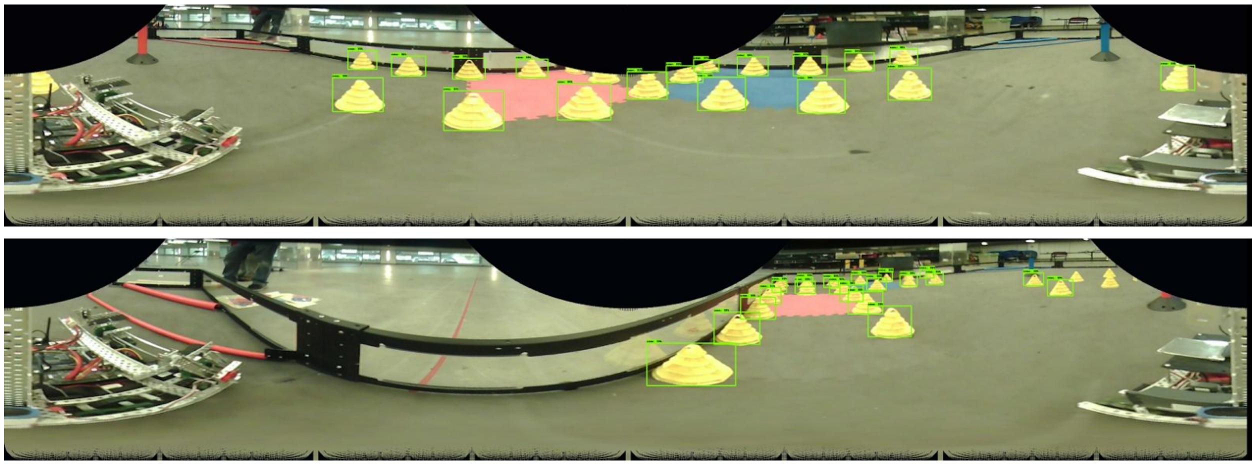


Figure 10: Examples of the trained object detection system

Using cross validation on our training set, we achieved a mean average precision (mAP) of 0.9, which while satisfactory, does not adequately speak to the real-world performance of the object detection system when running in real time on a mobile robot on novel data. While we believe the object detection system presented here to be useful and realistic, we ran into performance problems we did not have time to debug when implementing it on the robot during the duration of the MQP. We believe these to be issues of configuration mainly, but have not successfully proved that by the submission of this report.

3.4 - Strategic Planning

The design of our strategic planning system is based around a Q-Learning approach that records the average win probabilities in a random walk of the state space, guided by a reflex agent, as described in the background. We first built a discrete state simulator of the Vex robotics game, which clustered the cones into discrete field tiles, and represented the robots as agents occupying a full square and capable of moving 1 square in either direction.

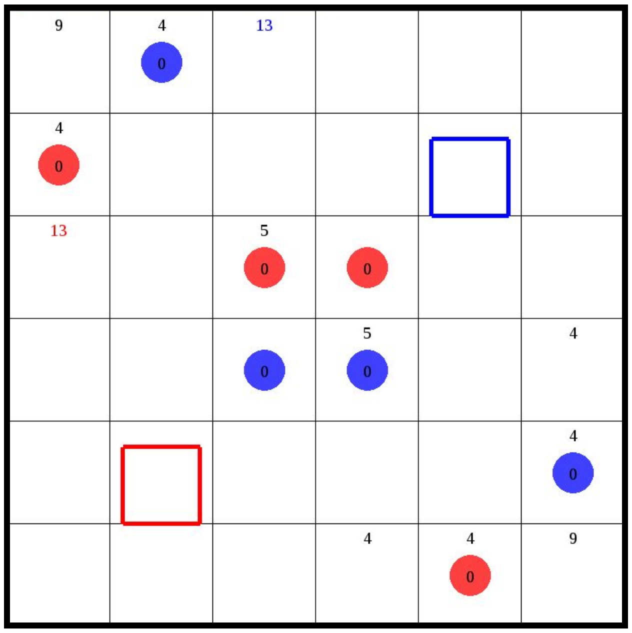
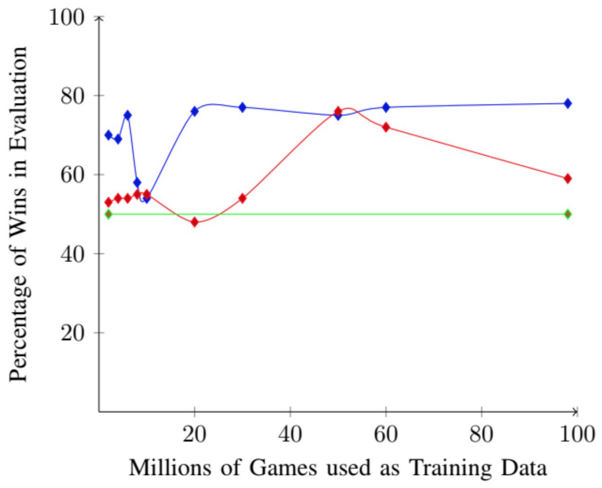


Figure 11: The simulated Vex game

After creating the simulation, we simulated 100 million games using the random epsilon method. However, we realized quickly that standard “greedy” Q-Learning, where you choose the highest expected value of all your possible actions, often chose high-value but uncertain actions, which made the efficacy of the system unpredictable and evaluations very noisy.



To compensate for this, we used a technique we call “Confidence-interval Q-learning”. We compute a confidence interval for each Q value, and then use the *lower* bound of the confidence interval when picking actions. In **Figure 12**, right, you can see the how the results of evaluating the trained agent against the naive reflex agent change with respect to the amount of training. Confidence-interval Q policy (blue) converges towards an 80% win rate, while standard Q-learning approaches a 75% win rate and then decreases.

4. Future Work

4.1 - Full System Integration

While we present prototypes and designs for what we believe to be the four major components of an autonomous robotic competitor, we did not unfortunately have the scope or time to completely integrate and test the entire robot as a whole. Although we hope our evaluations match reality, it is well known that theory and reality rarely line up. Accordingly, we expect that the scope of a project aiming to integrate these components and compete in an actual Vex match would be another full MQP to address the challenges and difficulties that arise during implementation.

4.2 - Temporal world-state estimation

One of our originals design ideas for the project was to have a world-state estimator, as proposed by Elfring et al. and implemented in the ros `wire` package. While we weren't able to include it in this project, we believe that putting it after the object detection system will give much better reliability, and resiliency to errors.

4.3 - Visual odometry

As mentioned in Section 3.1, our original design incorporated Semi-direct Visual Odometry specialized to run on a co-processor attached to the robot. While unexpected bugs in the software[2] prevented us from using it for this MQP, we believe it would be of great use to future MQPs looking for any type of odometry system, and especially future MQPs in this area.

4.4 - Monte-Carlo tree search

While Q-Learning was sufficient for the requirements of our project, with more time we would have liked to use it as a foundation for a more nuanced system using Monte-Carlo tree search, where you unroll randomly selected states until termination and then use it to

[2] https://github.com/uzh-rpg/rpg_svo_example/issues/27

train a policy evaluation function. While such a system would be more computationally expensive, we believe that it would be much more useful for predicting the long-term strategy of a match, and adapt much more easily to different opponent strategies. There are also intermediate improvements to be made, if MCTS turns out to be infeasible for computation on a mobile robot. Using Deep Q-Learning, where a neural network is trained to match the results of the Q value matrix, can often be shown to have benefits in generality (for appropriately chosen architectures and state representations).

References

J. Elfring, S. van den Dries, M.J.G. van de Molengraft, M. Steinbuch, *Semantic world modeling using probabilistic multiple hypothesis anchoring*, Robotics and Autonomous Systems, Volume 61, Issue 2, February 2013, Pages 95-105, (pdf)

Christian Forster, Zichao Zhang, Michael Gassner, Manuel Werlberger, Davide Scaramuzza; *SVO: Semi-Direct Visual Odometry for Monocular and Multi-Camera Systems*; IEEE Transactions on Robotics, Vol. 33, Issue 2, April 2017. http://ieeexplore.ieee.org/document/7782863/

Google Cartographer Project. "Google Cartographer." GitHub. October 13, 2017. Accessed October 13, 2017. https://github.com/googlecartographer/cartographer.

Huang, Jonathan, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. *Speed/accuracy trade-offs for modern convolutional object detectors*. April 25, 2017. Accessed October 13, 2017. https://arxiv.org/abs/1611.10012.

"RoboCupSoccer - Middle Size." RoboCup Middle Size League. Accessed October 13, 2017. http://www.robocup.org/leagues/6.

Rösmann C., Feiten W., Wösch T., Hoffmann F. and Bertram. T. “Trajectory modification considering dynamic constraints of autonomous robots”. Proc. 7th German Conference on Robotics, Germany, Munich, 2012, pp 74–79.

"Setup and Configuration of the Navigation Stack." Ros.org. Accessed October 13, 2017. http://wiki.ros.org/navigation/Tutorials/RobotSetup.

TEB Local Planner Authors. "Timed-Elastic-Bands Local Planner." GitHub. September 21, 2017. Accessed October 13, 2017. https://github.com/rst-tu-dortmund/teb_local_planner.

Tensorflow Authors. "Tensorflow Object Detection API" GitHub. October 04, 2017. Accessed October 13, 2017. https://github.com/tensorflow/models/tree/master/research/object_detection.

Vinyals, Oriol, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado Van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. "StarCraft II: A New Challenge for Reinforcement Learning." StarCraft II: A New Challenge for Reinforcement Learning. August 16, 2017. Accessed October 13, 2017. https://arxiv.org/abs/1708.04782.

Carl Vondrick, Donald Patterson, Deva Ramanan. "Efficiently Scaling Up Crowdsourced Video Annotation" International Journal of Computer Vision (IJCV). June 2012. https://github.com/cvondrick/vatic.

Appendix

A. Budget

	Per each	**Quantity**	**Total**		
Student Contribution	$250.00	3	$750.00		
Department Contribution	$250.00	3	$750.00		
NVidia Jetson	-$300.00	1	-$300.00		
Camera	-$45.00	2	-$90.00		
Hyoku Lidar	*-$1,000.00*	*0*	*$0.00*	(on loan)	
Harddrive	-$50.00	1	-$50.00		
VEX Hardware	*-$1,500.00*	*0*	*$0.00*	(robotics club hardware)	
Odriod XU4	-$65.00	1	-$65.00		
			$490.00	Total Cost of Project	

B. Code Listing

All code used in the completion of this project, including configuration files, can be found at the website https://github.com/asgard-mqp/ or (in a static snapshot) as part of the archival eCDR submission that accompanies this report. A brief overview of the most important repositories:

- https://github.com/asgard-mqp/robot_driver
  Contains the communication code used to control the Vex hardware from the Jetson side, as well as configuration files for the teb local planner
- https://github.com/asgard-mqp/brain_driver
  Not included in the eCDR submission due to a pending NDA, but contains the final code used to receive messages from the Jetson on the Vex side.
- https://github.com/asgard-mqp/launches
  contains the code and configuration, including launch files, used to run the required nodes on both the odroid and jetson
- https://github.com/asgard-mqp/AI-Final
  https://github.com/asgard-mqp/gym
  Contains the final code used for the strategic planning and simulation reinforcement learning.
- https://github.com/asgard-mqp/executor
  Contains the nodes for the high-level navigation (executor.py) and obstacle publishing used to execute the global strategic plan and transform it into concrete poses