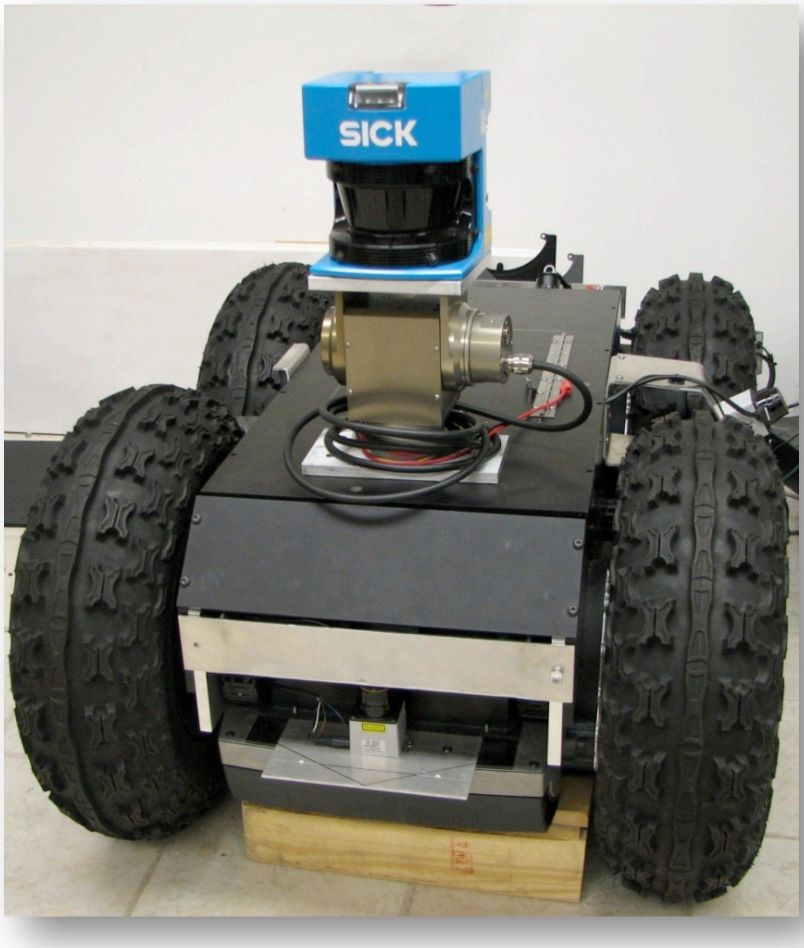
PENN STATE ROBOTICS CLUB

TELETRAAN I

IGVC 2009 Design Report

Rich Mattes, Jeremy Bridon, Anthony Cascone, Adam Brockett, Matt Poremba, Robin Pritz

June 5, 2009



Faculty Advisor Statement

I, Sean N. Brennan, certify that the design and development of Teletraan I has been significant, and that each student performing this work is a registered student. This work as part of a graduate class project and as an extracurricular project and represents a participation level equivalent to what would be awarded credit as a senior design project.

____

Sean N. Brennan, Department of Mechanical Engineering, Pennsylvania State University

Introduction

2008 was the first year the Penn State Robotics Club, in collaboration with Penn State's Intelligent Vehicles and Systems Group, participated in the Intelligent Ground Vehicle Competition. In that first year, we learned a lot about what did and did not work on our robot platform. We were also able to walk away with information on which strategies worked the best, and how best to prepare ourselves for the next competition. This year, we have an entirely new robot to enter into the IGVC. With help from Penn State's Networked Robotic Systems Lab, we are fielding a robot based on the Segway RMP400 platform. The robot was then modified to carry additional sensors, electrical power capacity, and computational power. The Segway RMP400 platform improves upon the drawbacks of last year's treaded tank design, and uses a robust commercially available drive system.

This year, we chose to rewrite most of our software algorithms from scratch. Although we are still using the Player/Stage platform, last year's algorithms were based on pre-written algorithms bundled with the Player package. Starting with pre-existing code did substantially decrease our development time, but it also adversely affected our robot's performance. Since these algorithms were not written from the ground up with our design needs in mind, they ran much slower than we would have liked. For this year, we threw out those elements and established a set of design criteria for our main software algorithms: map building and path planning. Then we worked over the course of the year to build each component from the ground up.

Design Process

For this year's IGVC, the focus was on improving the performance of our software over last year, and on finding a way to overcome the limitations of the treaded tank platform while staying within the resource constraints of the robotics club. The team was made up of six individuals working in their spare time over the course of the year to write new software, work on the platform, and debug our implementation. Below is a list of team members, their roles, and total time they have dedicated to the project.

Name	Department	Class	Responsibility	Hours Spent
Sean Brennan	MNE	Professor	Advisor	20
Rich Mattes	ESM	Senior	Team Leader	100
Jeremy Bridon	CSE	Junior	Lead Programmer	100
Anthony Cascone	MNE	Graduate	Platform Integration	100
Adam Brockett	CSE	Junior	Programming – Path Planning	30
Matt Poremba	CSE	Junior	Programming – Mapper	30
Robin Pritz	IE	Junior	Platform Integration	30

Platform Design

Failure Analysis

The main goal for the robot platform was to find a robust way to handle the shortcomings of the tank robot platform employed last year. Over the course of the competition, the tank treads frequently came off in the middle of a practice run. It was obvious that they could not handle the additional stress posed by the tall grass on the competition field. Further, the tank platform last year was unable to make it up the incline in the Autonomous Challenge without slipping back down. Given these drawbacks and the success of other teams' wheeled platforms, it was immediately obvious that we needed to transition away from treads to a wheeled robot platform.

The tank platform from last year also had several drawbacks in its power system. The system was run by four 18Ah sealed lead acid batteries, which were tucked away in the bottom layer of the robot. In order to access the batteries, the upper layers of the robot needed to be opened and folded back. This became problematic when we arrived at the competition last year, as we needed to shut down and disconnect many of our sensors in order to access the battery compartment. In order to stay running throughout the practice and competition stages, we needed to swap out all four batteries on a regular basis. The batteries only lasted about an hour and a half, which resulted in many battery swaps that took time away from practice. This year, we decided that we needed additional battery capacity to increase runtime, as well as a charging system that could be used without shutting down and disassembling the robot.

Another problem the tank faced last year was its limited top speed. We realized very quickly that if we were not able to run the robot at the maximum speed governed by the rules, we would not be competitive in the Navigation Challenge. The tank robot had a theoretical maximum speed of 5mph, but because of the tall grass and inclines, we observed an actual top speed around 2.5 to 3 mph. After the competition, several efforts were made to increase the overall top speed of the robot, including changing the gear ratio and mounting larger motors onto the platform. The increased gear ratio did result in a higher maximum speed, but also overheated the motors during testing. Mounting larger motors fixed the top speed issue, but the tank tracks came off much more frequently under the higher motor power. We decided that in light of these shortcomings, we would need a platform with a top speed much greater than the competition allowed, and regulate the speed accurately.

Because the tank platform was not able to maintain its top speed in competition, we found that during small maneuvers the robot would get stuck in place. The tank robot was equipped with quadrature encoders last year, but they were not used in competition. We realized that in order to assure we were moving as desired, we would have to have feedback from the wheels and used closed-loop velocity and yaw rate control. A platform with accurate wheel odometry was essential for this constraint.

Once we arrived at all of the design constraints for the platform's power and mobility, we had to decide how we were going to achieve these goals on our limited budget. Last year, the Robotics Club collaborated with the Intelligent Vehicles and Systems group and gained access to the tank platform and all of its sensors. This year, however, the Intelligent Vehicles and Systems Group are fielding their own entry into the IGVC, which puts large constraints on the accessibility of their equipment. Thanks to the Networked Robotic Systems Lab, we gained access to a Segway RMP400 platform and several sensors and computers to operate it with. This platform has a high capacity power system capable of running the vehicle for long periods of time. Further, the drive batteries only need 115VAC to charge, so we are able to charge the robot whenever we have access to a standard outlet. The Segway provides wheel odometry information and IMU measurements through a CAN interface, which is more than enough feedback to implement closed-loop velocity and yaw rate control. Finally, the top speed of the platform is advertised at 18mph, which is more than adequate for the needs of the IGVC competition. The RMP400 improves upon every area identified by last year's failure analysis.

Platform Modifications

After securing the RMP400 for use in the competition, several improvements needed to be made to the platform. The platform contains quite a bit of empty space under the top panel, but no good way to mount any auxiliary equipment. Several improvements would need to be made in order to equip the RMP400 with the instrumentation needed to complete the competition. A separate power source would have to be added to power all of the sensors, and needed to be able to provide up to 24VDC for our highest voltage sensors. A separate regulator also had to be installed to power the various equipment that ran on 5VDC, such as the Hokuyo lasers and USB hubs. Based on the power requirements, a 6A regulator was chosen to handle transient spikes on startup as well as sustained operation of all of the 5VDC equipment.

In addition to added power sources, the platform needed to have accessible mounting and storage locations for all of the sensors and equipment. One of our largest innovations for this year was to modify the RMP400 platform and add modular pull-out drawers in the front and rear. The front drawer is used to house electronics such as the USB junction for the Hokuyo lasers, the power breakout junction, and the RS422 to USB adapter used for the SICK laser. The rear drawer holds both laptop computers and another USB hub for the USB/CAN connections of each Segway module. These drawers allow all of the vital electronic components to be accessed instantly with no disassembly of the robot. Furthermore, all of the wires that run to the drawers are housed in a flexible plastic channel, which rolls out with the drawer and roll in underneath when the drawer is closed. This eliminates all possibility of binding, sticking, or pinching as a result of a moving drawer. Each drawer is held shut by two thumbscrews, which are permanently mounted in the drawer face. The screws are spring loaded, and pull the drawer closed once they begin to thread into the holes on the Segway platform. These springs also keep the screws from working their way loose due to vibration.

The top plate of the platform was also modified to allow access to the batteries. Several mounting holes for the SICK laser, and an access door for the 12V batteries, were cut into the top panel with a water jet. These allow the robot to be fully enclosed at all times, while still allowing access to the batteries in case they need to be swapped out.

Power and Electronics

The Segway RMP400 platform is a four wheeled, differential drive system with one motor per wheel. The construction of the RMP400 is unique, in that it contains two separate and independent RMP200 like two wheeled systems. That is to say, the front two wheels and the back two wheels each have their own independent drive modules, containing their own IMU, wheel odometry, motor controllers, and battery packs. Communication with the robot takes place as though one were communicating with two separate robots.

The RMP400 contains four 73.6V, 5.6Ah lithium-ion battery packs: two in the front module and two in the back. These batteries are sealed in the RMP400's individual drive modules. They can be charged by applying 115VAC to the charging ports: the charging regulators and circuitry are internal. However, these batteries can be charged while the Segway is on and running.

An additional power system is required to supply power to all of the sensors and computers on-board. We installed two 12V 72AH sealed lead acid batteries in the center of the robot. These batteries are connected in series to supply 24VDC power to the SICK laser. An additional 24VDC to 5VDC regulator, capable of supplying 6 amps of continuous power, was installed to supply power to the USB hubs and the Hokuyo lasers. Each 12VDC battery is charged individually from an external charger.

There are two Dell Latitude D430 laptops onboard, with optional high capacity battery packs. Each laptop has a standard charger attached to it, which provides power to the laptop when the robot is connected to 115VAC outlet power. While the robot is running, the laptops each run off of their own battery power. The laptops are capable of running for about 3 hours when the laptop displays and wireless radios are inactive. Each laptop has a solid state drive which improves performance with paging memory; this is especially needed with our mapping algorithm.

The complex charging needs of this platform require a unique charging solution. The RMP's rear panel is equipped with a large charging connector, with contacts for an 115VAC source, as well as two 12VDC charging sources. When connected, it supplies 115VAC to the Segway modules and to the laptop chargers. It also provides charging support for the two additional 12VDC batteries.

Sensors

Our robot is equipped with several laser range finders. We use a SICK LMS-200 as our primary range-finder; it is capable of providing measurements at up to an 8 meter range, over a 180 degree sweep, with 0.5 degree resolution. These laser scans happen at 37.5Hz. We have also equipped our robot with 3 Hokuyo URG-04 laser range finders: one in the front and one on each side. These units are much smaller and have a range of about 4m. They scan at 40 Hz, and have a 240 degree scan angle with a 0.36 degree resolution. The SICK laser is connected to the computers via a four port RS422 to USB adapter, while each Hokuyo is directly connected to the computers through USB.

The RMP400 platform contains two Segway modules, each with their own IMU and wheel odometry information. The wheel encoders are used to provide velocity information, accurate to about 0.003m/s, and wheel position information at 33215 counts per meter. The IMUs provide roll and pitch angle and rate and yaw rate. The units are also capable of providing motor torque and battery voltage information. The Segway refreshes its internal state data at 100Hz.

Our camera is a UniBrain Fire-I digital camera. It operates at a 640x480 pixel resolution, and transmits its data over the Firewire bus. The camera is capable of providing new images at 15Hz. We have fitted the camera with a 4.3mm focal length wide angle lens, to maximize the area able to be seen and processed by the robot.

Our GPS system is a NovAtel SPAN GPS/INS system. The OEM-IV GPS receiver is connected to a Honeywell HG1700 tactical-grade IMU. The system has an internal 14 state Kalman filter and is able to provide INS corrected GPS signals at a frequency of 100Hz. The system is accurate up to 10cm without DGPS correction.

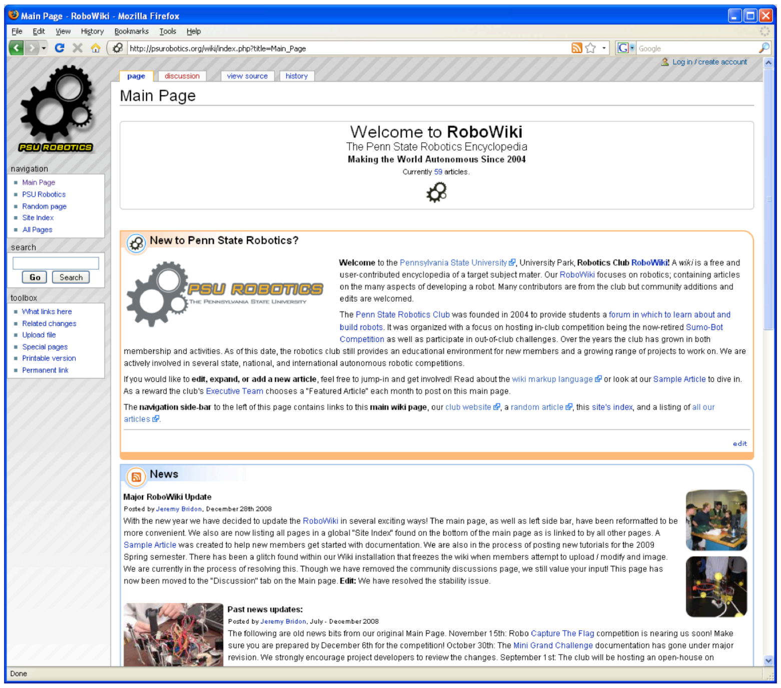
Software Overview

This year has had a heavy focus on an innovative, yet efficient, development approach to the software solution with measurable results. Our algorithmic solution to our system design includes two major components of map building and path planning. The process of map building is the generation, in real-time, of a global probability map that reflects real-world objects discovered by the vehicle. The second component, a path planner, is the logic to choose the most optimal path from our current position to a given destination. Within each of these two major components lay many smaller components such as sensor interfaces, image processing, and bitmap manipulation. Working together, these components work efficiently and choose reasonable paths through simulations and real-world trials.

Development Tools and Methods

Player/Stage is an advanced open-source robotics simulation and interface platform for UNIX and Unix-like operating systems. It provides both the tools to simulate, as well as communicate to, a robotics system. The platform is dynamic enough to allow one to create interfaces between a server (sensors, algorithms, databases) and client (controller logic). Player is the library that provides these standard interfaces as well as a standard protocol. Stage is the two-dimensional simulation platform. This platform is incredibly powerful in use with our iterative design pattern in which we rapidly test ideas and implementations, and keep components developed and tested in parallel. Player/Stage helped keep code and algorithms modular; so that they are easily transferred between platforms and projects.

Due to the academic nature of this project, we created several goals to help our software developers research different solutions, create a stable and efficient software implementation, and create resources for the robotics community. These goals were to focus on portability, simplicity, stability, efficiency, and openness. By keeping a simple design we allowed for stable and manageable code to be developed: critical to the large code base for such a system. We strove for high efficiency and performance for all code and algorithms, reinforced by peer-review sessions. We developed all code using open-source tools and freely publish our work to help further the robotics community.



Main Page

Welcome to **RoboWiki**
The Penn State Robotics Encyclopedia
Making the World Autonomous Since 2004
Currently 59 articles.

New to Penn State Robotics?

Welcome to the Pennsylvania State University, University Park, **Robotics Club RoboWiki!** A *wiki* is a free and user-contributed encyclopedia of a target subject mater. Our RoboWiki focuses on robotics; containing articles on the many aspects of developing a robot. Many contributors are from the club but community additions and edits are welcomed.

The Penn State Robotics Club was founded in 2004 to provide students a forum in which to learn about and build robots. It was organized with a focus on hosting in-club competition being the now-retired Sumo-Bot Competition as well as participate in out-of-club challenges. Over the years the club has grown in both membership and activities. As of this date, the robotics club still provides an educational environment for new members and a growing range of projects to work on. We are actively involved in several state, national, and international autonomous robotic competitions.

If you would like to **edit, expand, or add a new article**, feel free to jump-in and get involved! Read about the wiki markup language or look at our Sample Article to dive in. As a reward the club's Executive Team chooses a "Featured Article" each month to post on this main page.

The **navigation side-bar** to the left of this page contains links to this **main wiki page**, our club website, a random article, this site's index, and a listing of all our articles.

News

Major RoboWiki Update
Posted by Jeremy Bridon, December 28th 2008
With the new year we have decided to update the RoboWiki in several exciting ways! The main page, as well as left side bar, have been reformatted to be more convenient. We also are now listing all pages in a global "Site Index" found on the bottom of the main page as is linked to by all other pages. A Sample Article was created to help new members get started with documentation. We are also in the process of posting new tutorials for the 2009 Spring semester. There has been a glitch found within our Wiki installation that freezes the wiki when members attempt to upload / modify and image. We are currently in the process of resolving this. Though we have removed the community discussions page, we still value your input! This page has now been moved to the "Discussion" tab on the Main page. **Edit:** We have resolved the stability issue.

Past news updates:
Posted by Jeremy Bridon, July - December 2008
The following are old news bits from our original Main Page. November 15th: Robo Capture The Flag competition is nearing us soon! Make sure you are prepared by December 6th for the competition! October 30th: The Mini Grand Challenge documentation has gone under major revision. We strongly encourage project developers to review the changes. September 1st: The club will be hosting an open-house on

RoboWiki, our user-generated MediaWiki content manager

Using the free and open-source operating system Fedora 10 we were able to develop cost-efficient solutions that require no major licensing from 3rd party developers. Many other open-source tools were used for help in the development process. Subversion, or SVN, is a source-code management utility. This tool allowed us to program files in parallel, archive code, promote access to all developers, manage previous versions, and "roll-back" in the case of software failures. Through the practice of "continuous integration", we improved design and code quality. This process was applied through nightly builds of the code base in which any failure within the build process would result in an email containing all related error messages being sent to the developer team. In parallel to this, a nightly build of code documentation, generated through Doxygen, gave us access to each other's code documentation.

A MediaWiki website (dubbed "RoboWiki) was set up such that all software, hardware, and electronic designs were documented in a single accessible location. This allowed for constant design reviews and changes which were visible to the entire team at any point in time, solidifying our solution. Through this website, we hosted resolutions to common issues found in hardware or software development, aiding further our team and growing our knowledge base. A coding standard was documented to set consistent standards across all source files for better peer-review and development clarity. This MediaWiki can be found at http://psurobotics.org/wiki

General Software Solution

Our 2009 software solution is based on a two step feedback cycle, which is hardware independent, but with certain sensor needs. This software solution derives from our previous year's robot "Nitallion Battalion", though much of the code has been rewritten. The general solution builds a global map through different sensors, then attempts to navigate based on this map. This is repeated infinitely until the specified goal is reached. The map building process contains three steps: perception, processing, and fusion. The navigation process contains only two steps: planning and motion.

- Map building
  - Perception - Query sensors
  - Processing - Create a local representation of area
  - Fusion - Fuse the local map with a global map
- Navigation
  - Planning - Find the best possible path to the target waypoint on the global map
  - Motion - Translation to actual movement instructions

The real-world implementation is based on a multi-threaded process, with each thread completing different tasks. The Player abstraction layer automatically threads each individual sensor such that sensors are appropriately updated independent of our use of them. Each of our major algorithms and processes run in an independent, fully-encapsulated, driver that runs in independent threads, as managed by Player. The first driver thread, our mapping algorithm, has a target speed of one update per second, which varies based on memory resources, which is discussed in the global map section. We also set the map resolution, meaning the amount of real-world distance per element in our map, as 10cm. The second driver thread, the path planning algorithm, has a target speed of updating as fast as the mapping algorithm, but can be run faster. Our final thread is the main controller logic, referred to as the client, which communicates to these components making the correct high-level decisions. This main thread has a target processing speed of 100Hz.

Global Map Generation

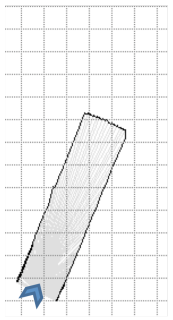
Our solution approach uses a probability map generated over time for use with a path planner. The sensor inputs are a front-mounted and two side-mounted laser range finders, vision data, and vehicle odometry. The laser data is used as a wide-sweep of collision data, which is helpful for obstacle detection around the vehicle. In the case of the autonomous challenge, the camera is used as a lane-detection system which treats the lanes as wall-like obstacles, passing laser range finder data back to this mapping algorithm.

Localization, by definition, is used to correctly place the vehicle and its data correctly into the global map. It is important to note that this algorithm was implemented as a Player driver, tested in Stage and later in the real-world.

The mapping algorithm takes three high-level steps per each cycle. Perception (Queries sensors), processing (Interprets sensor data), and fusion (Fuses local sensor data with a global map). The following algorithm is the formalized version of the above three steps with assumed inputs of laser scans, vehicle odometry, and outputs to a global probability map. 1. Initialize global probability map with all elements set to the "un traversed" state. 2. Initialize a local probability map with all elements set to zero (no obstacles) that is large enough to contain all laser range finder distances. 3. Query sensors. 4. Write out the collision data onto the local probability map; if the distance is greater than the maximum distance threshold, set that point to a probability of no collision (0). Else if the distance is within the maximum distance threshold and minimum distance threshold, apply an exponential decay function that takes the distance and returns a 0 to 1, representing the probability it is a finite obstacle. Else, distance is within the minimum distance threshold, so apply the point as a probability of full collision (1). These maximum and minimum distance thresholds are defined in a configuration file. 5. Merge the local map, based on vehicle odometry, into the appropriate location onto the global map. The function to merge the previous and new elements at a given point is based on an exponential average. 6. Go to step 2. In step 3 we attempt to query sensors, which might be too ambiguous for the vision sensor; its use is explained in detail in the following Lane Detection section.

$$\lambda = -\frac{\ln\left(\frac{1}{N}\right)}{U - L}$$

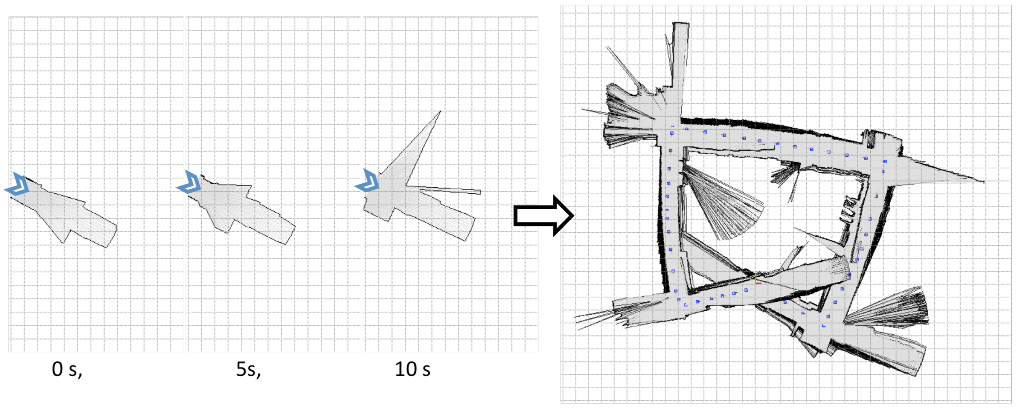
N = Maximum output value
L = Minimum threshold
U = Given distance



Exponential decay function (Lambda, given distance U) for local map generation. Note the further the object is from the center point, the less the point is defined as a finite obstacle; in this sample the robot is facing a hallway with no major obstacles, but the further away a wall is, the less it is a trusted point (e.g.: More white than dark).

$$\omega = (1 - ExpAvg)\beta + (ExpAvg)\alpha$$

Exponential combination function (Omega, where beta is the previous value and alpha is the new value) for local and global map merging. Note that "ExpAvg" is the exponential average constant, varying from 0 to 1.

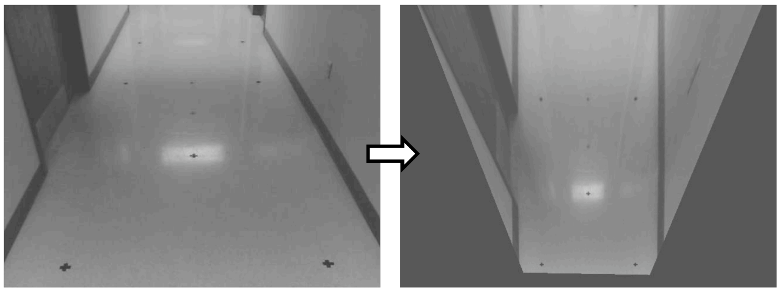


Several local maps generated over 10 seconds of movement, of the vehicle going down a hallway (towards the south-east corner), combined to create the bottom right of the global map's hallway. Note the global map's deformation due to localization inaccuracy, which is resolved with more accurate localization.

Lane Detection

Lane detection is critical to the autonomous challenge as it presents the need to have the vehicle stay within its lane, as defined by two lines. In our initial approach we combined a local planner with its own lane detection and a global planner independently. We found that when writing the high-level logic we created confusing situations in which either planner may attempt to do conflicting navigation. No known function was found to correctly combine the two separate outputs intelligently. Ultimately a combination of lane data combined directly into the global map for use by a single planner was needed. We chose to combine lane extraction with our global collision map, treating the lanes as possible collision points forcing the robot to treat it as any other obstacle, improving integration with the current code base. If the points that represent a lane are not continuous, we attempt to draw lines connecting the missing lane information, improving the visibility of a "wall-like" object.

Our lane extraction algorithm is based on taking the real-world image from a front-mounted camera, converting it to a binary map in which the background is subtracted (set as white) and the lane pixels, chosen by a range of colors, are set to black. The image is then warped via a perspective transform, making it seem as though it is a top-down view. This image represents an occupancy grid space of one meter by one meter. Such a transform can be superimposed into the global map, as the top-down perspectives make logical sense to merge. Based on the global map's merge function we can see the correct lane collision data with high accuracy within five to ten merges.



An application of the perspective transform; to the left is an image from the front-mounted camera, which is then transformed, outputting the image to the right. Noticed the clearly identifiable wall edge markings, as a straight line, and the clarity of the hall's width.

This combination of laser obstacle detection with lane detection creates a very clear global map in which our path planning algorithm can work within a lane and any obstacles.

Path Planner

The vehicle implements a single global path planner rather than a mix of a local and global planner; this is to prevent the complex run-time issue of choosing which planner is "more trustworthy". A path planner is a method of searching for the most optimal path from our current location to a final destination. We define optimal as the shortest, but safe, path without any collisions. After team reviews of several path-planning algorithms, we chose D*, a variant of the reverse Dijkstra search algorithm, as our path planner. This dynamic and efficient algorithm has the strength of not having to recalculate its entire internal data structures but only the local data that is associated with the changed global map, similar to Dijkstra's. D* is a real-time algorithm, in which its solution is reasonably optimal for our run-time resources constraint and is calculated fast enough to be usable without stopping the vehicle. It has a computational complexity of O (n^2) which is reasonable for our needs.

The D* algorithm changes information only when needed, which is optimal for our innovative global map approach. As our global map is only updated around the location of the vehicle, the internal data structures for D* also only update local data, without necessarily recalculating all data. Another powerful feature is its advantageous use of the probability map data, as well as the use of unknown/known states found in the map. This path planner never makes an assumption on unknown data points in the map, and attempts to only choose known safe areas for traversal if possible. In the case in which the robot places itself in a pocket, or is surround bed obstacles, the algorithm is smart enough to attempt a rotation to lead the robot out, rather than lead to a collision or attempt several three point turns. This path planner's target destination points are given by a high-level waypoint manager, described in the next section.

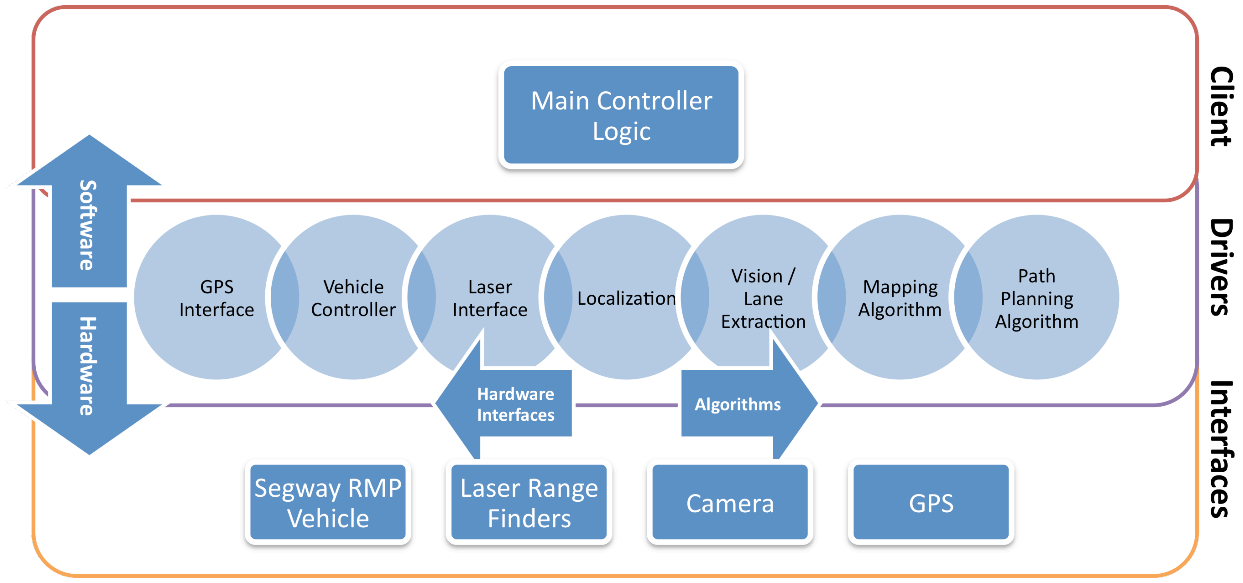
Waypoint Algorithm

A waypoint algorithm was applied as a high-level manager of the waypoints list, deciding when to give the next waypoint for the path planner's destination point. The robot runs an independent driver, through the Player standard, to manage these waypoints. Once the GPS informs this driver that the vehicle has reached a target waypoint, given a minimum distance, that waypoint is removed and the next is set as the vehicle's new destination. The first waypoint at the start of the vehicle deployment is set as the closest waypoint in the waypoint list.

In certain situations, navigation to one waypoint may actually lead the robot closer to other waypoints. In such a case we set that closer waypoint as the target waypoint. To prevent oscillations between waypoints in such a case, we hard-code a rule such that after two waypoint switches we force the vehicle to commit to either waypoints until the waypoint is reached. This forced-commitment resolves the oscillation issue.

System Integration

The Player/Stage platform provides three main levels of abstraction we use. The lowest level is the hardware abstraction and interface level. This layer provides the standard interface protocols between real-world hardware and virtualized hardware, which is needed to connect to the next level as well as provide a simulation bed for testing components. The next layer is the driver, or plug-in, layer that allows custom development of components. These components may vary in use, from an implementation of an algorithm to interfacing with new hardware. The top-most layer is the client library which provides high-level functions for easy interfacing with any algorithms or hardware components.



Our solution system includes several custom drivers and a single client-level main controller application. These drivers interface with hardware through the Player standard. The mapper, path planner, waypoint manager, GPS, and vision are custom drivers that communicate either in parallel, with hardware, and/or with the main controller. The laser driver and vehicle driver are a standard interface. The main control client only communicates with the path planner and vehicle controller, making the correct decisions on how to follow the given path. The rest of these components perform cross-communication in the background updating when appropriate.

The mapper driver requires localization, laser, and vision for correctly generating a global map. The path planner requires localization, the global map, and the waypoint manager, returning a bearing the vehicle should follow. The GPS driver requires the GPS hardware and returns GPS positions. The waypoint manager requires the GPS driver and returns, as appropriate, the best waypoint to move to. Vision is a drive that takes in pictures from the vehicle's front-mounted camera and returns, as collision data, the lanes found. The laser driver interfaces with the laser range finder devices and returns collision data.

Safety and Reliability

Our robot has a large focus on safety. The robotic platform is quite heavy and capable of traveling at great speed, so precautions must be taken to make sure the robot is not able to get away from the operators. Our control software for the Segways has an internal watchdog timer, and will stop the robot in the event that a new velocity command is not received every 400ms or so. This is to ensure that the robot will stop moving should the vehicle's control software routines hang or crash. The software also has an internal motor enable/disable function, and will not accept any new speed commands unless they are explicitly enabled. This prevents unsafe behavior on the vehicle's startup, and make sure the robot will not move until a program is run that explicitly tells it to.

Further, the robot is equipped with two hardware emergency-stop switches: one mounted on the robot and one that can be triggered wirelessly. These emergency stop circuits cut power from the individual Segway power units, disabling the motors immediately. The wireless unit has a range of 120 feet, and uses a UHF Auto-Roll system to eliminate the chance of rouge interference causing an unwanted emergency stop.

The electrical wires contained within the Segway are all properly and securely shielded and terminated, so that no unwanted shorts can occur. All of the wires are run through plastic channels, shielding them from the metal body of the robot, as well as from unwanted crimping or binding via prolonged contact with metal edges.

Performance and Efficiency

Platform Performance

The RMP400 is a high performance vehicle, with precision engineered self-contained locomotion. The drive system is borrowed from the 2-wheeled Segway RMP platforms, which are self-balancing. The self-balancing mode, however, consumes a lot of power due to the constant calculation and correction. The RMP400 is statically stable, so this mode is not needed and conserves battery life. The RMP400 is capable of travelling at a maximum speed of 18mph, which allows us to maintain the maximum permitted speed with plenty of headroom for hills, inclines, and tall grass.

The Segway platform increases the efficiency of our operation. The RMP400 is based off of a tried and tested, mass produced drive system. Using this off-the-shelf system, we are able to leverage the benefits of extensive testing and development time by the Segway Corporation. The end product is a reliable robotic platform that we do not have to spend time debugging or modifying as new problems crop up. As a result, we are able to spend most of our time working on the software development side, addressing the competition's algorithmic problems.

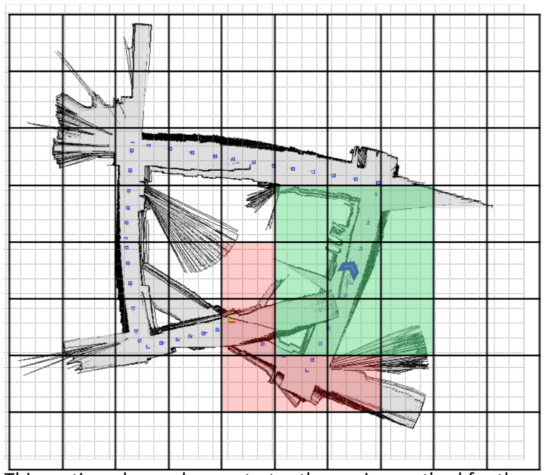
The modular drawer and charger design allows us to spend more time running the robot and less time deconstructing, should the battery run low. This makes our maintenance more efficient, as we can spend more time running the robot and less time maintaining it.

Should the robot be reproduced with all of its sensor equipment, the cost would be about $90,000.

Theoretical & Real-World Software Performance

The theoretical run-times expected for the two major components of mapping and path planning are O (n) and O (n^2) respectively. The map building is a linear process, though there are several major steps. These increase the computational run-time constant, but not the limiting behavior. In simulations it was found that map building, even with a high resolution of 5cm per map element and a large local map size of 8 by 8 meters, run-times were reasonable with several global map updates occurring within a second. The same performance was achieved with real-world deployment. The path planner had the expected theoretical run-time performing well within resource constraints in both simulations and real-world deployment. A valid path for a complex map, such as a circular hallway, was planned within 100 milliseconds. This was expected as D* only updates small internal data structures for the slight changes it receives from the global map, spreading work over smaller updates rather than re-planning on each update call. Though these results were satisfactory, several optimizations were applied to specific issues raised by the mapper and path planner.

The real-world implementation of the mapper is restricted by the computer's main memory. A global map cannot be of any large size because the system memory usage would be too high. Deploying on a 64-bit architecture with enough system memory is a valid solution, though naive as resource usage should be better managed. To solve this problem, a method of splitting the map into sections of finite sizes and paging them back and forth from main memory to secondary memory was used. Only "active" sections of the map are left in main memory. We define "active" as the sections that include the robot as well as sections that are adjacent to this location; in total nine sections are kept in main memory. A single section can vary to any size, though 10 meters by 10 meters worked efficiently. Any section that is not valid


This sectioned map demonstrates the paging method for the global map algorithm. Note the vehicle position and the highlighted green sections including, and adjacent to, its location; these represent currently loaded sections, with the non-colored sections paged out to secondary memory. The light red sections are those that have been recently released.

is said to be "inactive" and is saved as a serialized bitmap in the operating system's file system. As the vehicle moves, the mapper code will check for the location of the vehicle relative to these sections, and will write out "inactive" sections and may create or load new "active" sections if needed. This allows for the mapping algorithm to keep all previous data such that if the robot re-traverses known areas, it will load up correctly saved data of that area. In the worse-case scenario five sections are written out and five sections are loaded in. In such a case, this takes at most a full second, which is a reasonable solution especially since our software components are threaded.

We found that if the localization data is not accurate enough, it is possible that location drifting might occur, in which an accumulation of small errors lead to very large offsets in the vehicle's location in the global map. This is resolved as our GPS system is of high enough resolution the drift does not become an issue, and sensor self-correction helps with small offset errors.

As discussed in the relevant sections, we expected run-times to be all within reasonable limits for our resources. With the peer-reviews and simulations we were able to remove memory-leaks and resource mismanagement to improve stability over long periods of time, resolving the issue of major system slow-down experienced last year. As an added optimization, we deploy our software onto two separate computers: One as the main client managing logic and running the mapping and path planning drivers in the background. The second computer is the hardware interface (controller and sensor devices). Both computers communicate via a local on-board wireless LAN, which allows any number of "observer" computers within-range to log-in and observe the performance and actions of the system. The deployment of these components of software onto different machines increases the real-world performance significantly.