HANS - HUJI's Autonomous Navigation System

Yair Movshovitz, yairmov@cs.huji.ac.il, 040469710,
Keren Haas, keren_ha@cs.huji.ac.il, 015666209,
Dror Shalev, drorl01@cs.huji.ac.il, 032175994

Advisors:
Prof. Jeff Rosenschein[1], jeff@cs.huji.ac.il,
Nir Pochter[1], nirp@cs.huji.ac.il,
Dr. Zinovi Rabinovich[1], nomad@cs.huji.ac.il

September 15, 2008

[1]Hebrew University of Jerusalem Israel

Abstract

Building autonomous robots that can operate in various scenarios has long been a goal of research in Artificial Intelligence. Recent progress has been made on space exploration systems, and systems for autonomous driving. In line with this work, we present HANS, an autonomous mobile robot that navigates the Givat Ram campus of the Hebrew University. We have constructed a wheel-based platform that holds various sensors. Our system's hardware includes a GPS, compass, and digital wheel encoders for localizing the robot within the campus area. Sonar is used for fast obstacle avoidance. It also employs a video camera for vision-based path detection. HANS' software uses a wide variety of probabilistic methods and machine learning techniques, ranging from particle filters for robot localization to Gaussian Mixture Models and Expectation Maximization for its vision system. Global path planning is performed on a GPS coordinate database donated by MAPA Ltd., and local path planning is implemented using the A* algorithm on a local grid map constructed by the robot's sensors.

Introduction

Even before John McCarthy coined the term *Artificial Intelligence (A.I.)* in the famous 1956 Dartmouth Conference[1], researchers of A.I. have been interested in the notion of *Intelligent Agents* - Entities which observe and act rationally upon an environment. While this definition includes a broad spectrum of beings, such as web crawlers and software bidders in online auctions, the most intuitive way of perceiving an agent might be that of an autonomous mobile robot. A robot is a physical agent which uses different sensors (e.g. laser range finder, video camera) to observe its surroundings and actuators to modify the environment (e.g. motors, robotic arm).

There are various definitions for an *Autonomous Robot*, a straight forward definition is that of a robot which can perform its task without continuous human assistance. Such a definition raises the question - are modern day Industrial robot manipulators autonomous? Such robots, apart from being stationary, work in controlled environments. Very little changes in the environment that is not a direct outcome of the robot's actions. There is almost no variation in the situations the robot faces, and the entire environment was designed to enable smooth operation of the robot. These environments are called *Structured*. When we talk about autonomous robots in the rest of this report we consider a stricter definition:

Definition 1 *A robot is called* autonomous *if it can perform its task without continuous human assistance in an* unstructured *environment.*

To achieve this feat a robot must be able to overcome the inhereted uncertainty of working in the physical world. There are various aspects in which the uncertainty can be manifested:

The environment itself is unpredictable, it is dynamic and much more complex than those an industrial robot can expect to face, the situations and scenarios the robot may encounter are much harder to predict when designing the robot.

The robot's sensors & actuators are another cause of uncertainty. The sensors are limited in what they can perceive and are affected by their physical properties. For example, a sonar range finder trasmits a high frequency sound pulse in a defined direction and listens for the reflection of the pulse from objects in the surroundings. While the transmitted pulse moves through the air it broadens, after a couple of meters the pulse is very wide and therefore gives a very rough estimate of the location of objects in its path. The actuators are a source of uncertainty as well, the robot relies on them for moving around and modifing the environment but their actions are never precise, motors have acceleration and deceleration behaviors which differ between cold and hot motors. Breaking and slippage also affect the correctness of the robot's motion.

Another source of uncertainty comes from the way a robot models its world. The environment is too complex for the software to model it precisely so the robot uses generalizations, approximations, and heuristics to be able to hold a simplified version of the world in its memory and perform calculations on this model.

All these require that the robot be able to handle unknown situations thus making the task of building such an agent very demanding.

Related Work

Robotic development has been going on for more than a hundred years. As early as 1868 Zadoc P. Dederick published a patent for a "steam operated man"[2] and 20 years later Edison came out with

Figure 1: The top two finalists in the 2005 DARPA Grand Challenge. On the left is Sandstrom from Carnegie Mellon which arrived at second place. On the right is Stanley which won the race and 2M$ for Stanford.

a talking doll[3]. These, however, were very far from the modern idea of an autonomous robot. In 1986 Brooks[4] proposed a new paradigm of developing robots, which soon became quite popular, called *Behavior Based Robotics*. The idea was to view a robot as made up of different *behaviors* each performing a different task. Managing these behaviors are various *mediators* which handle cases in which two or more behaviors contradict each other (e.g. a planning behavior which wants the robot to move to a desired spot in front of it and an obstacle avoidance behavior which tells it to stop because an obstacle was detected also in front of it). In 2005 DARPA (Defense Advanced Research Projects Agency), the central research and development organization for the U.S Department of Defense held a robotic contest called "The Grand Challenge"[5] which was a 175 mile desert race for autonomous vehicles. Several teams completed the course and the Stanford robot, Stanley, won the race[6]. Figure 1 shows the top two competitors in the 2005 Grand Challenge. In recent years a new subfield of robotics called *Probabilistic Robotics[7]* has gained popularity. The core idea in Probabilistic Robotics is to employ probabilistic techniques for making decisions and representing information. This approach tackles head-on that inherited uncertainty in unstructured environments mentioned above and in many cases out performs any method which only takes into account a single "best guess" as to what should be done.

Our Project

Our project's goal was to build an autonomous mobile robot capable of navigating the Givaat Ram campus in a safe manner without any human assistance. In designing our robot we have used ideas both from Behavior Based Robotics and from Probabilistic Robotics. As we have set out to prove that autonomousy is achievable even within the scope of an undergraduate project we measure our success as the amount of different behaviors our robot can perform and the amount of integration between the different behaviors.

We have constructed a wheeled based robot that is able to drive on walkways and paved roads. The robot motors and its sensors are controlled using a Motorola microcontroller which resides in an integrated circuit we have built. Higher level abilities are achieved using an onboard laptop which performs all the vision tasks and complex statistical methods. We have written software modules for localizing the robot within the campus area, planning its route, detecting drivable areas using a video camera, and avoiding obstacles.

The rest of this report is organized as follows. An in-depth summery of the project's design, robot hardware, and software modules is provided in *Methods*. The *Results* section gives a detailed explanation of the performance of various behaviors, and the *Conclusions* section ends the report with a final discussion and possible future improvements.

Methods

Chassis & electronics

HANS was designed and built under the constraint that it should perform its mission while its total cost should stay under 1000$. HANS's body is made of PVC plates connected with PVC pipes. It has two front wheels powered by two $12v$ electrical engines allowing it to achieve a speed of $1km$ per hour depending on the environmental conditions. Power supply is provided by a 12V/4amp battery allowing one hour travel time. An external power supply can be connected for indoor testing.

HANS's sensor package consists of a GPS, wheel encoders (which can count the wheels' cycles), a digital compass, sonar range finder and a digital camera. A Motorola HC-11 microcontroller controls the engines, sonar, compass, and wheel encoders. The package is shown in Figure 2.

High level processing (decision making, image processing, localization, etc') is done by a laptop connected to the microcontroller by a serial cable.

After HANS was fully assembled and tested with most of its software modules, a commercial robot, Pioneer 2DX was found in one of the faculty storerooms. We used this opportunity to test the portability of our software and to implement more advanced obstacle avoidance modules using the Pioneer's 8-sonar array. The two robot platforms are displayed in Figure 3.

Architectural overview

When we came to design HANS's architecture and software, we had to deal with the issue of how to control an autonomous robot. Defiantly, the robot has multiple tasks that have to be carried out simultaneously while maintaining correct priorities among them (for example, the robot should not hit a wall while planning the global path to its goal).

First we defined how the robot should be controlled. In order to reduce the load on the laptop we decided that the low level control would be carried out by a microcontroller. The microcontroller is responsible for executing drive commands and operating the sonar, wheel encoders and compass. High level control is done by the laptop, which is responsible for the complete operation of the robot, and for the robot-human interaction.

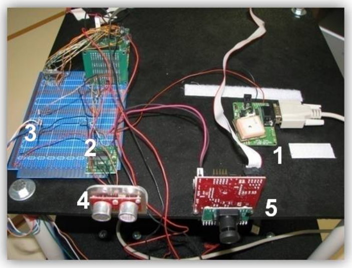


Figure 2: HANS's sensor package. (1) GPS. (2) Digital compass. (3) Wheel encoder circuit. (4) Sonar range finder. (5) Digital camera.

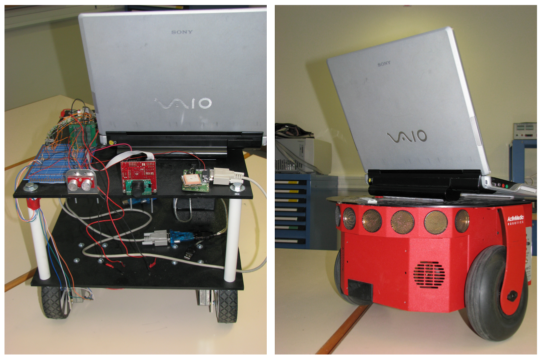


Figure 3: The two robot platforms used in this work. On the left - HANS based on the platform built by the team. On the right - HANS based on Pioneer 2DX.

We decided to implement HANS's control and operation based on the behavior based robotics paradigm[4], The main idea is that HANS's different capabilities and tasks are considered as human behaviors. As a human tries to reach its goal (e.g. go to work in the morning) it needs to constantly map its surroundings using his natural sensors, avoid hitting people or obstacles, while operating his body. In a similar manner, HANS needs to comprehend its surroundings using its sensors, and filter its different desires according to the current situation (reaching the final goal as quickly as possible might not be in line with not hitting the nearest wall), resulting in the correct command to its motors.

Figure 4 shows HANS's high level control in an abstract way. All inputs are colored pink, different behaviors, desires or data processing activities are colored gray and physical actions are colored green.

The main tasks the robot should carry out at all times are:

- Interpret sensor data.
- Plan a global path to its goal destination.
- Localize itself in the world.
- Build a local map of obstacles in its close vicinity.
- Plan a local path to reach the next waypoint in the global route.
- Find a drivable path in its surroundings.
- Execute local path.
- Avoid hitting obstacles of any kind.

Some of these behaviors might contradict each other from time to time, e.g. reaching the next waypoint according to the planned local path, might not be in line with the fact that a garbage can stands on the path. In such cases, modules that are called mediators, settle the disagreements and chose the most appropriate behavior (plan a new local path for example). Figure 14 in the Appendix section shows Hans's modules in more detail. Explanations of each module are also presented in the Appendix.

To implement the low level control we used the ICC11 IDE which allows programming in C and HC11 targeted compilation. High level control was developed using the player/stage robotic development framework which offers agreed interfaces, communication services, implemented utilities and a simulation environment. We used C++ for writing the different modules in the player/stage environment, JAVA for DB interface and MATLAB for probabilistic computations and image processing.

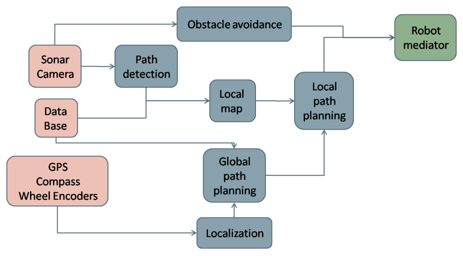


Figure 4: Abstraction of HANS's high level control diagram. Pink modules - inputs, gray modules - behaviors & data processing, green modules - physical actions.

State Estimation: Localization

The problem of determining a robot's position in the world (or pose, in robotics terms) is a central issue in robotics. Before the robot can make any intelligent plans it has to know where it is. In order to find its location the robot relies on its various sensors (GPS, Wheel encoders, etc'). However, simply using the sensors isn't enough to get good location estimates, the reason for this is that the sensors are noisy (skidding of the wheels can cause mistakes in the encoders count, the GPS has an inherent error of about $10m$, etc'). These small errors accumulate over time and quickly lead to false pose (Figure 5). Self localization techniques are used to compensate for these errors by comparing the acquired sensor data with the expected sensor readings. The sensors we used in our system were: GPS, compass, and wheel encoders.

The method we used to estimate the location of the robot in the world is called *Particle-Filtering*, which is a Bayesian estimation method. The main idea of Bayesian filtering is to estimate a probability density function (pdf) over the state space, conditioned on the sensor data. The state space, in our case, is the space of all GPS coordinates on earth. The Particle-Filter localization algorithm represents a robot's belief (of its location) by a set of weighted particles. The weight of each particle is the quality of that specific particle, in other words, the probability of that particle being the real location of the robot. After each movement of the robot, we perform several actions:

1. All the particles are modified according to the model of the previous stage – we know the pdf of the system at the previous time step and then we model the effect of the action to obtain a prior of the pdf at the current time (this step is the *prediction*).

2. Random noise is added to all the particles in order to simulate the effect of sensor noise.

3. Each particle's weight is re-calculated based on the latest sensory data (this is the *update* step). The new weight is the probability $P(Z_t|X_t)$ where X_t is the location represented by the particle and Z_t is the sensors' measurements.

4. The particles are re-sampled in a way that particles with small weights (low probability) are eliminated.

5. The estimation is obtained by the weighted sum of all the particles.

The process of *prediction* $\rightarrow$ *update* is repeated on each robot movement. The re-sampling step ensures that the algorithm converges. Both the accuracy and the efficiency of the algorithm are dependent on the number of particles, when using more particles we can get a better estimation, but the time to calculate each step increases. In our implementation we used an adaptive method which determines the number of particles according to the confidence the algorithm has of its location [22].

One of the difficulties of using this algorithm is to build the sensor model (i.e., to calculate $P(Z_t|X_t)$). We did this by modeling this probability as a Gaussian, where the parameters of the Gaussian were determined by doing field experiments.

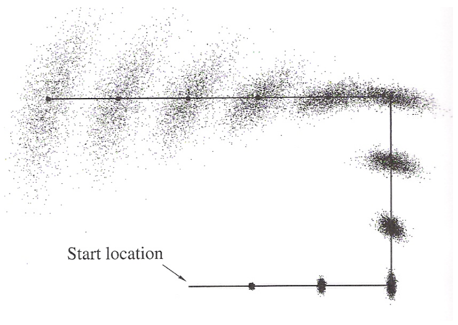


Figure 5: The problem of localization. Each dot is a possible robot location. As the robot moves the uncertainty in location grows as a result of the accumulative nature of the sensors' error.

Planning

The main goal of our robot is to be able to get to a given destination autonomously. In order to achieve this task it needs a way to plan a path to its destination. To plan the path from the start point, which is the current position as calculated by the localization module, to the destination, we used GIS (Geographic Information System) data donated to our project by MAPA Ltd. [23]. We used the geographic information of Givaat-Ram to build a GIS database using PostgreSQL DB [24]. The path planning task was divided into two modules: global path planning and local path planning.

Global Path Planning

The role of the global path planning module is to calculate a path between the starting position and the destination, based only on the GIS data stored in our DB. From that information, we construct a weighted graph in which nodes represent crossroads in the campus, edges represent paths, and each edge's weight represents the length of road between two crossroads. Using the Dijkstra algorithm we find the shortest path in the graph, which correlates to the shortest route to the destination point. Figure 6 illustrates the global path planning process.

Local Path Planning

The global path planner outputs a list of GPS coordinates the robot should pass on its way to the destination point. These *checkpoints* are typically distant $20 - 30$ meters from each other. To calculate the path between the points outputted by the global path planning module, a local path planning technique is required. The local planner considers the location of roads, buildings, obstacles etc'. For that purpose we build a map of the local area of the robot using data gathered from the database, sonar, and vision system (the localization algorithm is used to determine the area to be mapped). The local map is rebuilt after every 10-20 meters. The optimal local path is found using the A* algorithm which is a fast search algorithm that uses heuristics in order to speed up the search. We needed a faster search strategy than Dijkstra in the local path planner because, unlike the global scenario, the local path is recalculated each time an unmapped obstacle is detected which blocks areas that were considered free just a moment ago.

Vision

As our robot can only drive on walkways and paved paths, we needed a way to distinguish between drivable and non-drivable areas. HANS is suited with a front facing video camera and we decided that it will be used as our main tool for this task.

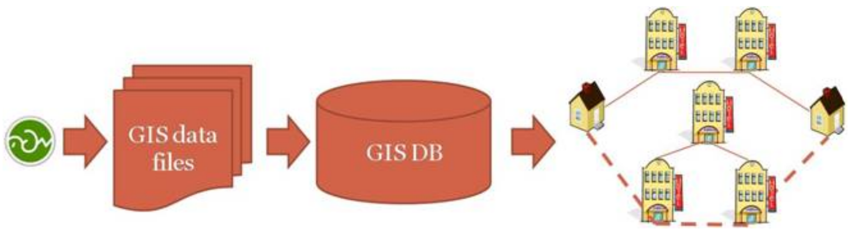


Figure 6: Global Path planning process. GIS data donated by MAPA Ltd. is transformed into a GIS Database. A graph representing the campus is constructed from the DB. The Dijkstra algorithm is used to find the shortest path in the graph.

The notion of using computer vision to detect roads is not new, as early as 1988 Thorpe et. al.[8] presented a color based system for automatic vehicle road following. In 1995 Pomerleau[9] suggested another road following vehicle which broke down the problem into three stages 1) sampling of the image, 2) determining the road curvature, and 3) determining the lateral offset of the vehicle relative to the lane center. Since then there has been much research on this problem and many ideas have been tested ([11, 13, 12, 10, 6] just to name a few). However this problem is difficult and still remains an active area of research.

The problem our robot faces is even more difficult than that of road following vehicle. Vehicles can expect to only drive on paved roads, which always have similar color and texture. HANS needs to detect paved roads, stoned walkways and paths. These can come at any possible color! Furthermore, the robot can switch between different types of roads in a single run. Lighting conditions also play a factor in vision based road detection as the robot can be used in different times of day when the color and intensity of the light varies.

With these difficulties in mind, we designed and implemented a vision system for HANS. Figure 7 illustrates the process of detecting road areas within an image (that is, segmenting the image into two parts - road and non-road.). Our system has four main steps, we elaborate in detail on each step below:

1. Sampling pixels.
2. Learning models.
3. Building image graph.
4. Segmenting the image using the graph.

Sampling the pixels

The first step in detecting drivable areas is to build two data sets of pixels on which learning can be done. One data set holds pixels that will be used to learn a *Road Model*, and the second data set's pixels will be used to learn a *Non-Road Model*. There are two types of strategies for building such data sets, offline - where the data set is built in advance, and online - where the data set is built dynamically during the robot's regular activities. We have implemented this stage of the vision process in both ways:

- Offline - A database of road images from around the campus was gathered, we wrote an application which allows a user to manually define areas which are part of a road/path, and used it to compile the two data sets.
- Online - On each new frame that arrives as input to the algorithm we heuristically define two rectangled areas, one (at the bottom of the image) which indicates a good probability for being part of the road, and the other (at the top) which indicates a good probability to be part of the background. Figure 8 shows one input frame with the described selections. We use the

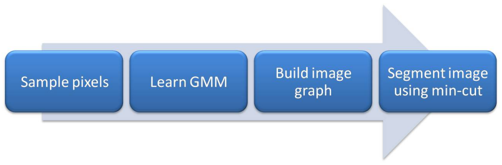


Figure 7: Vision system flow diagram. First we sample pixels to obtain a data set in which we can perform the learning process. Then we learn the parameters of two Gaussian Mixture Models using the data set. We use the GMMs to assign road and non road probabilities for each pixel. After that we build the image graph using the calculated probabilities, and finally we segment the image by cutting the graph.

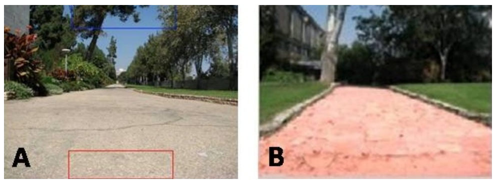


Figure 8: (A) Heuristically building data sets. The red (bottom) rectangle is used as the road data set. The blue (top) rectangle is used as the non-road data set for this single frame. (B) A frame after detecting the road. The area overlaid with red is the *road*, everything else is the *background*.

pixels from the bottom rectangle as the road data set, and the ones from the top rectangle as the non-road data set.

In our tests we have found that both methods work adequately and give a good assessment as to where is the road (see the *Results* section). The offline version is slightly faster and more accurate, but is less robust as it is dependent on the images in its database and so can only detect paths which are similar to those that are represented in it. The online version is more general, capable of coping with unexpected types of roads/paths but, as mentioned, a little less accurate. As the two methods both provided good results we decided to keep both, and the decision on which to use is now up to the user, which can change the behavior as a parameter of the system.

Learning the Gaussian Mixture Models

The second step of our algorithm requires us to extract relevant data from the two data sets of pixels gathered in the previous stage. In our case this means learning the *typical* color of the road/background and save it in a way that will enable us to quickly and easily make decisions about the classification of unknown pixels.

All objects in the image (the road being no exception) are comprised of a variety of colors distributed around a main color, for example the sky in Figure 8 holds a number of shades of the color blue, i.e. each pixel in the sky area has an RGB value which is close to blue. This leads us towards modeling the road/background as a Gaussian centered around the typical color of the object. This may work for very simple types of objects, but more realistic objects are usually comprised from a number of different colors. So a more general way of modeling the road/background is by using a Gaussian Mixture Model, i.e., a set of gaussians, each with a weight factor that indicates how much the object leans towards that color. Formally, Let I_r be the 3 dimensional vector of the color values of pixel r in the current frame. We model the road globally using a Gaussian Mixture Model:

$$P_{road}(I_r) = \sum_{k=1}^{L} w_k \cdot P_G(I_r|\mu_k, \sigma_k)$$

Where P_G is a normal distribution and w_k, μ_k, σ_k are its weight, mean and variance. L is the number of components in the GMM. That is, the probability of a pixel I_r being part of the road area is the (weighted) sum of probabilities for it to be part of the color gaussians comprising the road. Similarly we define $P_{bg}(I_r)$ as the probability of a pixel r being part of the background. We now need to learn the GMM's parameters, namely w_k, μ_k, σ_k and L. A popular way for learning Gaussian Mixture Models is by using the EM algorithm[14]. This gives us the weights, means, and variances but a decision needs to be made about the maximum number of gaussians. We have found that road objects usually contain about 4 different colors, and background areas are slightly more complex. To be on the safe side we used a version of EM that uses component annealing, L is initially set to a relatively large value (we used $L = 10$) and trivial components are discarded during the learning.

Building the image graph and segmenting the image

The road extraction problem can be viewed as a labeling procedure, where each pixel r should be labeled as either *road (1)* or *background (0)* . In [15] it was shown that the labeling problem can be solved using min-cut if it is stated as an energy minimization problem. To solve the problem we used a method similar to [17], we created a graph as follows: we defined a source and a sink (road and background) and for each pixel r in the image we defined a node N_r. Each node is connected to the source and to the sink. There is also a connection between each two nodes that represent neighboring pixels in the image (4 neighbors). For each pixel r we defined a *color term*:

$$ColorTerm(x_r) = \begin{cases} -logP_{road}(I_r) & x_r = 0 \\ -logP_{bg}(I_r) & x_r = 1 \end{cases}$$

The color term is the cost of assigning r with the label x_r. In min-cut notation this is the cost of removing the edge between N_r and the source/sink. For each two adjacent pixels $r; s$ we also defined a *contrast term*:

$$ContrastTerm(x_r, x_s) = |x_r - x_s| \cdot e^{-\beta \cdot d_{r,s}}$$

where $d_{r,s} = \|I_r - I_s\|^2$ is the L_2 norm of the color differences and $\beta = \mathrm{E}[\|I_r - I_s\|^2]^{-1}$. The contrast term is the cost of assigning r with the label x_r and s with x_s. This is the cost of performing the cut in the graph between N_r and N_s. The desired labeling can now be achieved by cutting the graph in two, each connected component corresponds to one of the labels. To cut the graph we used the implementation of the min-cut algorithm in [16]. Figure 8 shows the outcome of the segmentation procedure on a typical frame (see the *Results* section for an in-depth analysis).

After the road and background areas have been selected accurately the output needs to be forwarded to the local planning algorithm, this requires the image to be projected to a coordinate system of a bottom facing camera in order for distances to be measured correctly. This stage has not been implemented yet, it involves using consecutive frames from the video input and the approximated speed of the robot to estimate distance in the picture and then to project the image according to these distances.

Obstacle avoidance

As the robot carries sensitive hardware that must be protected, it needs to guarantee that no collisions occure with objects in the environment. Such collisions may happen if the output of the local planning algorithm or the vision algorithm produce erroneous decisions as a consequence of bad sensor readings. Even in a perfect world, where the sensors contain no noise, and both the local planning and path detection algorithms work with 100% accuracy, there is always a risk of sudden appearance of objects due to the dynamic nature of the campus environment - people (or cats) may cross the path of the robot. To guarantee a risk free ride, we have built a failsafe mechanism that uses the robot's sonar range finder for fast obstacle avoidance. As the sonar works at a much higher frequency than both the vision and planning algorithms, it provides a fast way of detecting both static and dynamic obstacles.

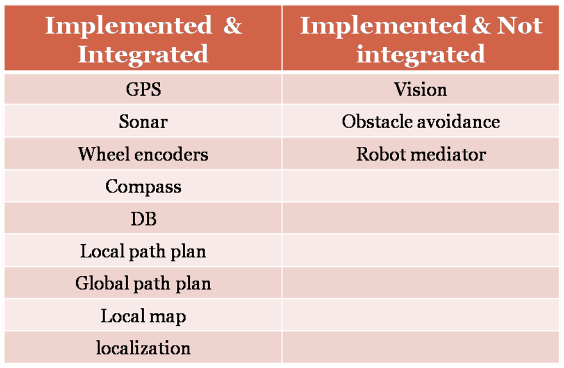
There are numerous works on the subject of obstacle avoidance using range finding sensors, [19, 18] are two popular methods, however, these methods are aimed for indoor robot obstacle avoidance which is more complex than outdoor scenarios (there are more obstacles and they occupy a larger percentage of the area) and so they are computationally "heavier" than is actually needed in our case, especially when the robot has already planned a safe route to the closest check point and only needs the avoidance behavior as a failsafe.

For our build-it-yourself version of HANS, which has only a single front facing sonar, we have implemented a simple obstacle detection mechanism which halts the robot when it gets too close to an object. Trying to make more complex decisions according to this single sensor will undoubtedly lead to undesired behavior of the robot, but there is no absolute need for making such actions - if the obstacle is dynamic (e.g. a cat) it will probably be on its way in a moment or two, so, if the robot can wait patiently, the road will be cleared for it without it needing to act. If however the object is static and for some reason has not been mapped until now, it will be mapped in the following few seconds by the vision system, and the local path will then be re-calculated.

For the HANS version that runs on the PIONEER 2DX, which has an array of 8 sonar range finders, we have implemented a slightly more complex behavior, in which, when the robot encounters an obstacle, it turns to the "path of least resistance", i.e., turns towards the direction in which sonar readings indicate the longest free distance, and moves in that direction. Again, after a moment or two, the local path is re-calculated and the new route can be followed. The benefit of this technique is that the new route is likely to pass through the stated "path of least resistance" and so in most cases the robot will seamlessly avoid the obstacle.

Current status

HANS's original plan turned out to be pretty ambitious for a three person team over 3 semesters. After consulting with our advisors, we decided to continue the development of each module independently and leave their integration as the last step. Following this plan we managed to implement (and fully test) all the planned modules and have integrated most of them. However, several of the modules have not been integrated yet. For example, the vision module is implemented and tested, but still doesn't update the local map. We estimate we have completed 80% of the original plan. The table in Figure 9 presents all the modules of the project and shows the status of each one. On July 30th the HANS project won an excellence award as one of the three top engineering projects of 2008 in the Hebrew University of Jerusalem.



Implemented & Integrated	Implemented & Not integrated
GPS	Vision
Sonar	Obstacle avoidance
Wheel encoders	Robot mediator
Compass	
DB	
Local path plan	
Global path plan	
Local map	
localization	

Figure 9: Current status of software modules.

Results

localization

In order to test our Localization technique we ran two types of tests: Simulated tests and field tests. We wanted to be sure that our algorithm improves the position estimate of the robot when compared with only using the sensor data.

In order to be able to compare the performance of localizing using data from only one sensor (GPS, odometry etc') versus localizing using the Particle Filter method, we developed a simulation program that creates a random path and the (noisy) sensor readings that correspond to it. At each run of the simulation, the robot moves 300 steps. Figure 10 (A & B) shows a localization result of a simulation run when only considering one type of sensor at a time. Each sensor has its own shortcomings. The GPS, while producing readings that are guaranteed to be no further than $10m$ from the correct position, has no memory of past readings. It therefore has a tendency of creating non-continuous paths. The odometry (i.e., wheel encoders) reading, on the other hand, produce a smooth path, but its errors accumulate and cause a drift from the ground truth data. Figure 10 (C) shows a simulated localization run when using the Particle Filter algorithm. It is visible that this technique benefits from the data provided by both sensors and overcomes their shortcomings. It produces a smooth path and its error does not accumulate.

We quantified the consistency of our algorithm, by running the simulation 100 times on different paths. We defined an error measure as the distance between the true path and the estimated one. From the results in Figure 11 it can be seen that the Particle Filter technique not only reduces the mean error but also produces consistent results.

After getting satisfying results from the simulation we tested our algorithm in the "real world". As no ground truth could be gathered in these tests (even the GIS data base has an approximate error of $1.8m$) our tests were mainly visual sanity tests. We inserted the localization data into Google Earth and verified that the estimated path corresponds to the path the robot traversed in the field test.

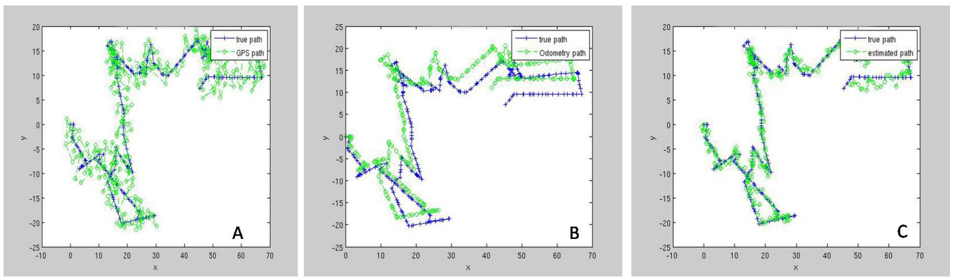


Figure 10: Localization results from simulation runs, blue crosses indicate the true path while green circles show the estimated path. (A) GPS estimate. (B) Odometry estimate. (C) Particle Filter estimation.

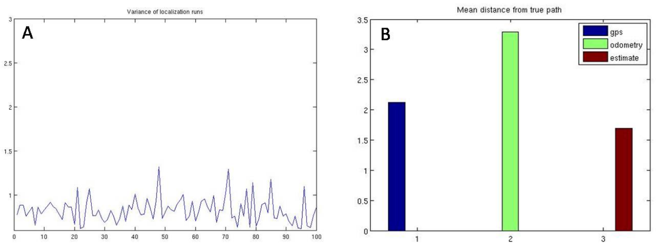


Figure 11: Localization statistics when using the Particle Filter algorithm. (A) The variance of the error in each of the simulation runs. Notice that the algorithm produces a stable error. (B) Comparison of the average error over 100 simulation runs between using a single sensor (blue and green bars) and using a Particle Filter estimate (red bar).

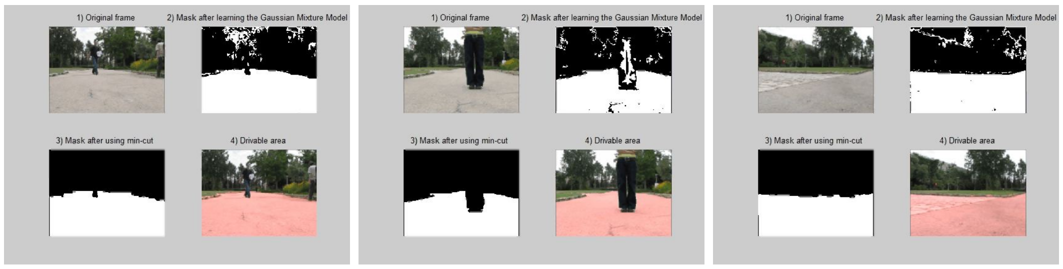

1) Original frame
2) Mask after learning the Gaussian Mixture Model
3) Mask after using min-cut
4) Drivable area

Figure 12: Typical frames from a test run. The robot can expect to have people and other dynamic objects in the frame. Notice the change of paths in the right most image.
(1) Original frame. (2) The labeling mask when using maximum likelihood from the GMMs. (3) The labeling mask when also using min-cut. (4) The labeling of road areas (in red) is overlaid upon the original frame.

Vision

While some modules, such as the localization process, can be analyzed statistically, the vision module has no ground truth to which we can compare our results, nor can we run simulation of it. Thus we present our results as best we can using sample frames from our test runs. The reader is encouraged to view our full length videos of the vision module[1]. The road finding process produced good results on various types of sequences, mainly on such where the road colors are uniformly distributed. Figure 12 shows the results of the vision module on a number of typical frames from various test runs. Notice how the algorithm handles static/dynamic obstacles in the frame, and also how it copes well with a change in color and texture when switching between paths. When experimenting we found that, while the system is quite robust for small to medium illumination changes, errors in labeling can occur when shadows are involved. In the *Conclusions* section we talk about a possible solution to this problem.

[1]For supplementary material visit http://sites.google.com/site/hanssupplementarymaterial/

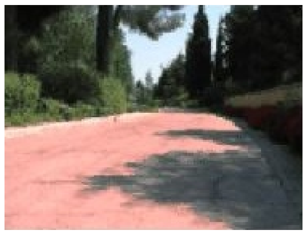


Figure 13: Shadows can cause mistakes in the road (marked in red) finding process.

Obstacle avoidance

Due to time limitations, and since the obstacle avoidance module was planned and used mainly as a safety measure, its testing was constructed as a series of sanity tests. Both HANS I and HANS II were tested indoors and outdoors in various scenarios (environments containing walls, chairs, boxes, people, etc') for their ability to stop moving when encountering an obstacle, without crashing into it. Both showed 100% success in avoiding the obstacles, even in scenarios when the obstacle was suddenly placed in their path. Since HANS II has an 8-sonar array, the obstacle avoidance module implemented for it was more complex, based on the "path of least resistance" approach as was described in the *Methods* section. We didn't quantitatively test whether the direction chosen by the robot after encountering an obstacle is always the path with the longest distance to obstacles. Nonetheless, HANS II was able to cross each obstacle maze it faced during the sanity tests without hitting any obstacles. Video demonstrations of HANS I simple obstacle avoidance and HANS II "path of least resistance" are available in the supplementary material web page.

Conclusions

We have set out to show that robotic research has reached a point where the algorithms and paradigms which are popular today are robust enough to allow the building of autonomous physical agents without using sophisticated and expensive, specialized, components. All our sensors, motors, and other electronic parts have been bought as off-the-shelf items, which are available to anyone with an internet connection. The total cost of the materials was approx. 1000$, on top of which we have used a laptop that was available from one of our advisors.

We have built a mobile robotic platform which is able to drive on walkways and paved paths carrying an onboard laptop. Our software is able to localize the robot in the campus area, plan paths to desired locations, find drivable areas using a video camera, and avoid obstacles while performing these tasks.

We believe that the modules we have presented in this report demonstrate how to create building blocks of an autonomous robot. When combining these blocks into a single, real-time, system we get a robot which has all the capabilities we need for autonomous navigation in a campus area. As was mentioned in the *Current Status* section, most of the modules have already been integrated with each other and work, in real-time, together.

Future Work

There are a number of directions in which future work can improve the capabilities of our system. First, the integration of all modules into the system should be done.

In our localization module more data can be extracted from the GPS (such as the number of satellites it is connected to) and the importance given to its readings can be modified accordingly. Another improvement can be to build an image database of buildings around the campus, and using them to assist in location estimation, thus creating a sort of *visual memory* of the robot, this memory can also be updated during actual robot runs.

In the vision system, work can be done on both accuracy and speed. As of this day the system works at approx. 4 frames per second, this is acceptable as the robot itself is not very fast, but if the frame rate would be increased the robot will be able to detect obstacles and other environment changes in a safer manner. As was discussed in the *Results* section, shadows sometimes cause mistakes in the path detection process. There are a number of techniques for shadow detection and removal ([20] is a good example) and these can be used as a pre process of the path finding module.

When detecting obstacles our local path planner simply re-calculates the path from scratch using A*. A faster approach would be to only modify the current plan, as is done in the D* light algorithm[21].

Acknowledgments

First we would like to thank our advisors Jeff, Nir, & Zinovi for their guidance throughout the process of designing and building our robot. The team would also like to express their sincere gratitude to Doron First for the time and effort spent helping us with various hardware issues. Lastly we would like to thank MAPA Ltd. for their donation of the GIS data of the Givat Ram campus.

Bibliography

[1] http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html

[2] http://davidbuckley.net/DB/HistoryMakers/1868DederickSteamMan.htm

[3] http://davidbuckley.net/DB/HistoryMakers/1890EdisonTalkingDoll.htm

[4] Brooks, R. A robust layered control system for a mobile robot

[5] http://www.darpa.mil/grandchallenge05/index.html

[6] Thrun et al. Stanley, the robot that won the DARPA Grand Challenge.

[7] Sebastian Thrun, Wolfram Burgard, Dieter Fox. Probabilistic Robotics.

[8] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer. Vision and Navigation for the Carnegie-Mellon Navlab.

[9] Dan Pomerleau. RALPH: rapidly adapting lateral position handler.

[10] Ohno and Tsubouchi, A mobile robot campus walkway following with daylight-change-proof walkway color image segmentation.

[11] 12. Ernst D. Dickmanns. Vehicle Capable of Dynamic Vision.

[12] Guilherme N. DeSouza and Avinash C. Kak. Vision for mobile robot navigation: A survey.

[13] 14. Massimo Bertozzi, Alberto Broggi, Alessandra Fascioli. Vision-based intelligent vehicles: State of the art and perspectives.

[14] A. P. Dempster, N. M. Laird and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm.

[15] Yuri Boykov and Vladimir Kolmogorov. An ex- perimental comparison of min-cut/max- ow algo- rithms for energy minimization in vision.

[16] Y. Boykov and M. Pi. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images.

[17] Jian Sun, Weiwei Zhang, Xiaoou Tang and Heung-Yeung Shum. Background Cut.

[18] Ulrich, I. and Borenstein, J., 1998, VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots.

[19] J. Minguez, L. Montano. Nearness Diagram Navigation (ND): Collision Avoidance in Troublesome Scenarios.

[20] Graham D. Finlayson, Mark S. Drew, and Cheng Lu. Intrinsic Images by Entropy Minimization.

[21] Sven Koenig, Maxim Likhachev. D* Lite.

[22] Vandi Verma, Sebastian Thrun, Reid Simmons. Variable Resolution Particle Filter.

[23] http://www.gisrael.co.il/

[24] http://www.postgresql.org/

Appendix

The following list provides detailed explanations for all the modules shown in Figure 14:

1. Motor – Gets either an angle to turn or a distance to travel and moves the robot accordingly.
2. Sonar – Provides distance to closest object in front of the robot.
3. Camera – Outputs frames.
4. Compass – Outputs the direction the robot is facing.
5. GPS – Outputs a GPS reading, but in grid coordinates.
6. Dist – Returns distance traveled in a straight line since last reading.
7. DB – Holds GIS data and returns intersections/road data upon request.
8. Anti Collide – Failsafe mechanism. Stops the robot if there is an object less then X cm away.
9. Lane Process – Gets a raw image data and calculates probabilities for each pixel to be a road/non road pixel. Outputs the probability matrix.
10. Obstacle + edge detect – Looks at a probability matrix and tries to find objects, road edges etc' in it.
11. Grid map – Holds a grid of local area. Each cell has a probability to be drivable/non drivable. Provides this grid upon request.
12. Localize – Gets data from compass, GPS, dist and point 2 point and calculates a vector of possible poses and their probability.
13. Global path plan – Gets data from the DB and localize. Builds a graph of the campus area and finds the shortest path to the requested end point. Outputs a list of checkpoints (in grid coordinates) to the checkpoint holder.
14. Global checkpoint holder – Holds a list of checkpoints the robot needs to go through in order to reach its destination (a list of (x,y) coordinates).
15. Replan – Tells global path plan to recalculate the global path if a certain route proves to be impassible.
16. Obstacle/grid mediator – (just mediator in the image) gets data from obstacle detect and from grid, decide which one to believe and updates the grid.
17. Local path plan – Plans a route in the grid from a starting point to an end point. Outputs a list of (x,y) to go through, or no route if none is passable.
18. Local tracker – gets the list of (x,y) and tracks the advancement of the robot according to the list.

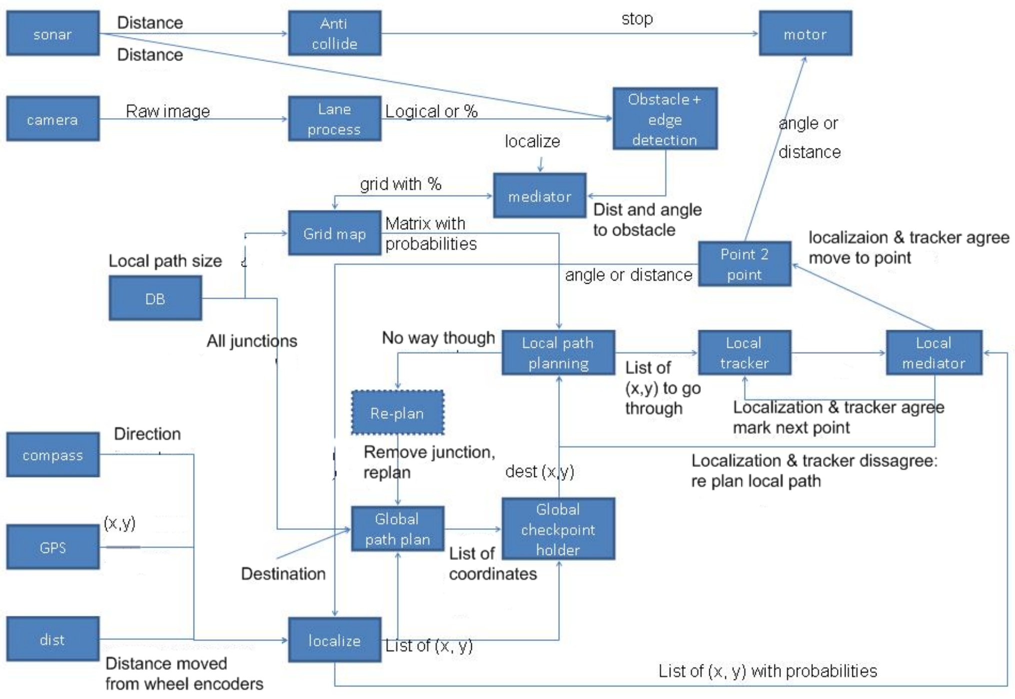


Figure 14: HANS's full architectural design.

19. Local mediator – gets possible poses from the localize module and the local tracker (localize provides a list of top probability poses). If they agree (the (x,y) in local tracker is one of the (x,y) in the list from localize) tells point 2 point to move the robot, else tells local tracker to recalculate the route (it believes the localizing module more than the tracker).

20. Point 2 point – Calculates the angle to turn or distance to move.