Robotics and Autonomous Driving

Hannah Kerner, Alan Kuntz, Jeffrey Ichnowski, and Michael North

May 4, 2015

To help realize the vision of autonomously driving vehicles, we look to the field of robotics to answer the question of how to get to our destination efficiently while avoiding obstacles. The foremost question one must ask before planning a route, and continuously while en route, is where are we now? This is a non-trivial problem, given the inaccuracies and uncertainties induced by sensors such as GPS, odometry, and computer vision. Once we have a method to localize the car, we can then plan a route--that is, compute a motion plan. Computing a motion plan is also complicated by an ever changing world with many rules of the road and an absolute safety-critical requirement of not injuring or killing anyone while travelling. As the localization and planning software is thus a key component of a safety-critical system, we must have some way of validating or proving the correctness of its operation--this includes both logical correctness, and temporal correctness. In simple terms, this means that our computation of a motion plan must correctly find a solution, and do so within a provable time bound. In this paper, we first describe how to localize a car, we then explore a case study of how a research vehicle autonomously drives in an urban environment, and then we look to how to make the motion planning of the car run within a hard real-time system with provable correctness. Finally, we describe a robotic platform on which such a system could be built.

Vehicle Localization (H. Kerner)

When planning for the autonomous navigation of a robot, one of the earliest and most fundamental challenges one faces is accurately measuring the location of the robot at all times. For an autonomous robot to follow a road, it has to first know where the road is, and where it is with respect to the road [Levinson 2007]. Robot motion planning algorithms require a robot's *pose*, which consists of the robot's location and orientation. While orientation is relatively simple to determine since the system can directly measure this using a compass, measuring location is not so straightforward. The classical approach in robotics is simply to try to estimate the robot's *state* based on the information available (in this paper, state is defined as the robot's position and velocity). "Pretend that there is no longer any uncertainty in state, but prove (or hope) that the resulting plan works under reasonable estimation error" [LaValle 2006]. While most motion planners do try to account for uncertainty in a variety of ways, they are generally mitigating effects from several different sources of uncertainty, such as dynamic environments and actuation, not specifically the robot's location. Researchers instead tend to account for uncertainty in localization heuristically and on a case-by-case basis using methods such as enforcement of minimum clearance bounds (i.e. "safety bubbles") in planning.

Today, vehicles not equipped with GPS are few and far between, but the accuracy and availability of GPS prevent it from being the sole means of measuring location for autonomous navigation of automobiles. In their 2014 GPS Performance Analysis Report, the FAA reported horizontal accuracies ranging from 2.192 to 8.465 meters (depending on the city) with 95% confidence and horizontal accuracies with 99.99% confidence ranging from 4.595 to 16.851 meters [FAA 2014]. This is clearly not accurate enough, nor is it held to an accurate enough standard by the GPS Standard Positioning Service (SPS) Positioning Standard, for motion planners to safely assume the location of a vehicle to be known [DoD 2008].

Navigation applications for autonomous driving and advanced driver assistance systems (ADAS) can be greatly improved by localization with centimeter accuracy. Not only does an autonomous robot need to know with certainty where it is with respect to a road if it wants to follow that road, but if an autonomous robot wants to follow a lane on a road, it also needs to know precisely its location with respect to the lane on the road it is following. From Levinson et al., "For an autonomous robot to stay in a lane, the localization requirements are in the order of decimeters." The level of accuracy needed for this and other autonomous and ADAS applications cannot reliably be achieved using GPS-based inertial guidance systems, particularly in urban areas [Levinson 2007]. In addition, the very availability of GPS is a less than reliable assumption in urban environments where unexpected GPS-affected regions occur semi-frequently, such as parking garages or "street canyons" (roads surrounded by buildings on both sides so as to create a canyon-like environment). The assumption of a known location and that "the resulting plan works under reasonable estimation error" [LaValle 2006] also becomes especially risky in urban as opposed to rural environments where narrow passages and sharp turns are more prevalent.

Researchers have used myriad algorithms and techniques for improving localization accuracy. "In recent years, one of the most widely accepted approaches in experimental mobile robotics is to use sampling-based techniques to directly compute and estimate the probabilistic [information states]" [LaValle 2006]. This observation seems to hold true today, as the majority of papers on mobile robot localization, and in particular those specific to automobiles, use sampling-based methods for accurately determining the vehicle's location. Common themes across techniques in recent years for precision vehicle localization are Kalman filters, road-matching or map-based algorithms, and particle filters. In this survey we will explore a map-based method for localization using a particle filter developed by Levinson et al. at the Stanford Artificial Intelligence Laboratory. This method achieved a localization

accuracy that was much greater than GPS-IMU-odometry based methods and accurate enough for autonomous navigation and ADAS applications [Levinson 2007].

In 2007, Levinson et al. proposed augmenting the GPS-IMU-odometry based system (a standard feature in automobiles today) with a LIDAR ranging sensor to first learn a detailed map of the environment and then use real-time ranging data to localize the vehicle relative to this map. Maps are acquired by a vehicle equipped with a state-of-the-art inertial navigation system with GPS and three down-facing SICK laser range finders (LIDAR sensors) pointing to the left, right, and rear of the vehicle. These LIDAR sensors return a range to a sampling of points on the ground as well as a measure of infrared reflectivity. The resulting map is an overview of the road surface in the infrared spectrum including a variety of textures in the environment that might be useful for localization, such as lane markings (which have a higher infrared reflectivity than the road) or grass on the road edges, as shown below in Figure 2 from Levinson et al. 2007. A modification of the GraphSLAM algorithm is used to compute road maps from the sensed information that retain only measurements coinciding with the flat ground plane, having eliminated vertical objects such as guardrails as well as non-static features such as moving or parked cars. Because the rectangular area acquired by the laser scans are decomposed into a square grid and squares are only saved if they contain data, losslessly compressed grid images occupy about 10MB for each mile of road at a 5-cm pixel resolution [Levinson 2007].

Figure 2. Visualization of the scanning process: the LIDAR scanner acquires range data *and* infrared ground reflectivity. The resulting maps therefore are 3-D infrared images of the ground reflectivity. Notice that lane markings have much higher reflectivity than pavement.

The localization step is done in real-time using a particle filter (also called a Monte-Carlo localizer) that maintains a three-dimensional pose vector of x, y, and yaw (assuming the odometric measurements are sufficiently accurate for roll and pitch). The particle filter determines which road in the map the vehicle is on by analyzing ranging and GPS data, correlates the infrared reflectivity measured by the LIDAR in real-time with corresponding measurements in the map, then tracks the vehicle's location by projecting particles forward through time using the velocity outputs from of the vehicle's wheel odometry and IMU-GPS velocity measurements. To mitigate complications in correlating infrared reflectivities resulting from variations in the reflectivity of the same feature under different weather conditions, brightness and standard deviation were normalized for each range scan and for matching map stripes.

Using a filter with 200-300 particles and a 5-cm pixel resolution, Levinson et al. found that the localization system "very reliably tracked the location of the vehicle, with relative accuracy of about 10 cm". They also showed the system was "robust to dynamic and hilly environments" so long as "the local road surface remains approximately laterally planar in the neighborhood of the vehicle". Furthermore, their system was able to successfully localize the vehicle with manageable error even with GPS turned off, using only odometry and steering angle with LIDAR. Although the horizontal

error using this method was occasionally as large as 30 cm, experiments using only GPS for localization consistently failed within meters to accurately localize the vehicle. “Using the localization technique described here, the vehicle was able to follow [a fixed reference] trajectory on ten out of ten attempts without error; never did [this] method fail to provide sufficient localization accuracy” [Levinson 2007].

Autonomous Motion Planning for an Automotive System: A case study of BOSS, the DARPA Urban Challenge winner. (A. Kuntz)

Many approaches have been taken in autonomous vehicle design. Each approach varies in many ways in almost every aspect. No two designs, systems, or algorithms are the same. With such a large variance in the approaches taken to solve this problem, it may be difficult to place approaches into a single framework to analyze. As such, it may be valuable to examine a specific instance of a successful autonomous vehicle. With such an examination, it may be possible to draw inspiration not for specific ways to solve the problem, but in the type of problems and solutions to be addressed in future systems design.

One such success story is that of BOSS, an entry in the DARPA Urban Challenge resulting from a collaboration between Carnegie Mellon University (CMU), General Motors Research and Development, Caterpillar, Continental AG, and Intel Research. A survey on many aspects of the BOSS system was published by its developers in the Journal of Field Robotics [Urmson 2008]. BOSS was able not only to qualify first but to win the challenge. While BOSS utilized effective and complex techniques in all aspects of its operation, including sensing and localization, this paper is going to focus on a survey of the algorithms which BOSS utilized to plan its actions and motions.

BOSS utilized a three layer planning system. This system consisted of a mission planning layer, a behavioral reasoning layer, and motion planning layer to effectively navigate urban environments. The three layers act as a hierarchy to plan coarse actions to fine controls. While each layer is distinct in the types of planning it is performing, there is a large level of communication between the layers facilitating effective planning at every resolution.

Mission Planning

The most coarse resolution of planning performed by BOSS is the mission planning layer. This layer is responsible for determining what roads the vehicle needs to traverse to navigate from a start to a goal location. In the case of the Urban Challenge, the state of the environment was given to the team in the form of a road network definition file. The team used this data to encode the environment into a graph--a set of vertices and directed edges--which corresponds to the connectivity of the road network. In this case, the team represented driving lanes as edges in this graph. Each edge was given a weight, which was based on many factors, including distance, time to traverse, and some notion of the complexity of the environment around the edge.

In addition to encoding a static representation of the connectivity of the environment, the mission planning layer updates this graph dynamically as it gets new information. This is done through the detection of "blockages", a notion that an edge has become infeasible for traversal. The mission planning layer handled two categories of blockages. The first, physical blockages, correspond to sensed obstacles which make the road impassable. The second, virtual blockages, are reported if the motion planning layer does not find a feasible path through an edge, for whatever reason. In each case, a large cost is added to the edge and a new route is planned. Because blockages are inherently transient--i.e. road crews may remove a broken down vehicle--this cost is also transient. In the case of the physical

blockages it decays exponentially over time, and in the case of virtual blockages it is removed completely each time the vehicle arrives at a checkpoint

The problem of solving for a route from a start location to a goal location is then reduced to a simple graph search on this data structure. This route then serves as a baseline for the subsequent planning layer, the behavioral reasoning unit.

Behavioral Reasoning Unit

The behavioral reasoning unit makes decisions about how to go about executing the plan defined by the mission planner. It does this through executing lane changes, safely traversing intersections and yielding points, lane and distance keeping, and error recovery. The behavioral reasoning unit functions similar to a finite state machine, making decisions based on the current context under which the vehicle is operating. These contexts take the form of things like *lane driving* or *intersection handling*.

Intersections and Yielding

The primary mechanism by which the behavioral reasoning unit handles intersections and yielding is by determining precedence, or which vehicle has the right of way in intersections. This is done with knowledge of traffic heuristics, e.g. the vehicle to the right has the precedence if it is otherwise ambiguous. Additionally, the vehicle estimates the order in which vehicles arrive at an intersection. It does this through drawing occupancy polygons at the entrances of intersections, and keeping track of what vehicles have entered the polygons and in which order. It is doing this to determine when it has precedence itself so that it can execute an action to proceed through the intersection.

It handles yielding onto a road which does not have stops in a similar fashion. Estimating the velocity of vehicles on the road onto which the vehicle is merging, it constructs large polygons based on travel speeds in both directions of travel. If BOSS senses a vehicle in either of the polygons then it

does not have precedence. In this way it is able to yield to vehicles that are traveling on the destination road. Once it determines it has precedence, it will plan its action onto the road.

Lane Driving

The behavioral reasoning unit also handles the specifics of driving on a road. This takes the form of planning lane changes and distance keeping with respect to vehicles in front of it.

If a lane change is desirable, the behavioral reasoning unit will determine if it is feasible to change lanes. This feasibility is based on whether it is able to find a plan to change lanes that maintains proper spacing with surrounding vehicles while also constraining its velocity in the adjacent lane. It first determines if it has time to execute the lane change based on the vehicle in front of it and approaching checkpoints. If so, it must decide how to fit into the vehicles in the target lane. It considers merging in front of and behind each of the adjacent vehicles (for a total of k + 1 options if there are k relevant vehicles in the lane).

The behavioral reasoning unit utilizes the motion planner for determining specific plans for executing desired behaviors. In this way, the motion planner unit and the behavioral reasoning unit work in concert.

Motion Planning

The specifics of motion planning are broken up into two large scenarios, and the algorithms utilized for each are different. The two scenarios considered are on-road and zone navigation. On-road navigation is distinguished by the natural path defined by the environment. This takes the form of the centerline of a lane, or the curve between two lanes in an intersection. The zone navigation is used for motion planning in unstructured environments, such as parking lots or during error recovery.

On-Road Navigation

Trajectory generation is used to track the desired path on a road. The trajectory generation algorithm used specifically can be found in [Howard 2007]. The algorithm parameterizes the trajectory space using spline parameters for curvature, and linear velocity profile parameters. With the set of trajectories now described by a vector of parameters, an optimization problem is formed where the value being optimized is the error of the trajectory at the boundary values. An iterative optimization is then performed in the trajectory space.

This trajectory generation algorithm is used to produce a set of trajectories that follow the center line of the road to a varying degree. Specifically, the end point a set distance away is perturbed in the direction tangent to the road and trajectories are planned to these discrete perturbations. This allows a different trajectory to be selected--one which was precomputed--if an obstacle is detected on the ideal trajectory.

Zone Navigation

In the zone navigation setting, the goal is no longer a point on the centerline of a road, but instead a pose within the zone. To plan the motion from a current vehicle state to the desired pose in the zone, a lattice based planner is used to search on a discrete set of possible local maneuvers, considering state of position, orientation, and velocity. The resolution of the discretization is higher when considering states close to the vehicle and lower when considering distant states. The lattice based planner used is an instance of the anytime replanning algorithm Anytime D* [Likhachev 2005].

Starting with a suboptimal--but quickly computed--motion plan, Anytime D* then uses the remaining computation time to iteratively improve the motion plan. Notably, there is a provable upper bound on how suboptimal the current plan is through the course of the iterations. From a high level, the planner searches backwards from the goal pose to the current pose, considering discrete controllable

state transitions. Also, because it is retaining the graph on which it is searching as it executes the plan, it is able to replan if sensing provides information which renders the current motion plan infeasible.

This zone navigation planning algorithm generates a trajectory to the goal pose. During execution, the same perturbation technique described above is also used during trajectory execution.

Zone navigation is used both in unstructured environments such as parking lots, and as error recovery during other phases of driving, in anomalous situations where other paths are infeasible. For more details on the motion planning system, the reader may also refer to [Ferguson 2008].

The three tiered system worked well for planning the actions of BOSS. It is important to note that communication between the three systems is critical to successful execution. In addition to many eloquent algorithmic solutions, as is the case in many real world systems, a large number of heuristic solutions were also applied.

Real-Time Motion Planning for Autonomous Driving (J. Ichnowski)

The motion planning system in an autonomous vehicle is a safety-critical system--that is, incorrect operation could result in injury or death to the occupant(s) or humans in the vicinity of the vehicle. Motion planning is the process of computing a sequence of actions that takes a robot (e.g., car) from an initial configuration to a goal configuration while avoiding obstacles and obeying kinematic constraints. In this case of a car, the configuration minimally refers to its position and orientation, though for planning purposes may often include additional important attributes, such as velocity. In order to guarantee correct operation, both logical and temporal correctness of the motion planning algorithms must be verified. To guarantee temporal correctness, most schedulability tests require that the tasks in the system be characterized by their phase, period, worst-case execution time (WCET), and relative deadline. The difficulty with real-time motion planning is that motion planning is a

PSPACE-hard problem [Reif 1979], with many algorithms having time-complexity that is exponential in the dimension of the robot's state space, and an indefinite (possibly unlimited) run time--making a WCET time analysis problematic at best. In other words, motion planning in the general case is a difficult problem, even without a hard real-time constraint. This section describes approaches to making motion planning algorithms for autonomous vehicles run within a hard real-time system. It begins by describing the motion planning problem, then general approaches to motion planning that could be adapted to a real-time system, it then covers WCET analysis of a hard-real-time motion planner, and concludes with a case-study of three finalists in the DARPA Urban Challenge.

Motion planning is the computation of a sequence of actions that takes a robot from an initial configuration to a goal configuration while avoiding obstacles and obeying kinematic constraints. In the general case, a robot's configuration can be represented as a vector of size equal to the degrees of freedom of the robot. For example, a humanoid robot with 25 movable joints, has 25 degrees of freedom, and its configuration is representable by a vector of 25 dimensions. A car's configuration is minimally its position (x,y) and orientation (θ) on a ground plane, and is thus 3-dimensional vector. A car is controlled in a 2-dimensional space: steering and acceleration/deceleration. This leads to a non-holonomic constraint, which put simply means the car cannot instantaneously move to the side.

Motion Planning with Hard Real-Time Constraints

While many approaches to motion planning consider an offline pre-planning process in a static environment, there are a few approaches that take into account a fixed compute time limit. Even for planners that work best in offline pre-planning, there are adaptations that are possible to make the motion planning operate in a hard real-time environment. In this section we cover general approaches to real-time motion planning through the use of: (1) potential fields that allow robots, even with high degrees of freedom, to avoid obstacles with a real-time computation; (2) precomputed roadmaps that

allow fast execution of motion-planning queries; (3) anytime planning that can be stopped and queried anytime for a best path; and (4) grid- and lattice-based planners operated by discretizing space and performing a graph search.

Khatib, in [Khatib 1986], proposed a motion planner that avoids obstacles through the use of potential fields, and did it in such a way as to provide a compute time limit. Obstacles in this method have a potential field that "pushes" the robot away, with a stronger force as the robot gets closer. Since the robot and obstacles operate in a 3D workspace, but the robot is controlled via a multidimensional joint vector, the method uses Jacobians to translates the "push" from the workspace into the robot's configuration space (e.g., joint movements). Potential fields suffer from problem of getting stuck in local minima, however this method has a boundable WCET as it applies forces to the robot's bounded number of joints.

Roadmap planners, such as the PRM[1] [Kavraki 1996], precompute the connectivity of the obstacle-free space in a static environment. Once computed, the robot performs a graph-search on the roadmap to find a path from any initial configuration to any goal configuration in the environment. With an appropriately limited roadmap size, it is conceivable that a WCET analysis would provide a computation bound that is amenable to a hard real-time motion planner. The issue with this type of approach is that the WCET is highly dependent on the roadmap, and thus an analysis may be difficult to perform in a general case. Another issue with the roadmap-based approach, is that it assumes a static environment, and thus requires an additional layer in order to react to changes.

Anytime planners, such as the asymptotically-feasible RRT[2] [LaValle 1998], and asymptotically optimal RRT* [Karaman 2011], compute a single motion plan and may be stopped and queried at anytime for their best motion plan. The RRT and RRT* planners work generating a random

[1] Probabilistic Road Map
[2] Rapidly-exploring Randomized Tree

robotic configuration one sample at a time, and connecting the sample to a tree. provided it and its connection to the tree are obstacle-free and obey kinematic constraints. RRT is asymptotically-feasible--with infinite time, it will find a solution with probability 1.0 (provided a solution exists). RRT* is asymptotically-optimal, in that with infinite time, it will not only find the solution, but the solution will have the shortest path as defined by a cost function. While it is fairly straight-forward to bound the computation time by limiting the number of samples considered (as we will see in a following section), these motion planners randomly sample configurations and thus have a chance of not finding a solution within a time-bound--even in relatively unobstructed workspaces.

Grid- and lattice-based planning works by discretizing the configuration space or action-space of a robot, and performing a graph search (e.g., A*) on the resulting discretization. To handle dynamic, changing environments, algorithms such as D* [Stentz 1994], D* lite [Koenig 2002], and Anytime D* [Likhachev 2005], perform an A*-like search and keep information from one graph search to the next. They speed up subsequent motion planning queries in the presence of small changes in the environment. By searching on a bounded discretized space these algorithms can ostensibly be easily analyzed for WCET. To the best of our knowledge, there has not been a published WCET analysis for a planner based upon a discretized A*-like approach. While in practice these algorithms can work well [Likhachev 2009] as pathological worst-case motion planning problems are rare, the WCET from analysis of an discretization-based approach may prove them incompatible with a schedulable (period > WCET) and responsive (period < 1 second) system. That is, in order to make the WCET fast enough to respond quickly to environment changes, the discretization may have to be on an unusably low resolution grid.

Hard Real-Time RRT with WCET Analysis

Walker, in his dissertation [Walker 2011], created a Hard-Real-Time Rapidly-exploring Randomized Tree motion planner, analyzed its WCET, and validated the timing analysis through experiments. The basic outline of the method was alluded to in the previous section and is presented in pseudo-code below. In each periodically released job, it builds a planning tree via RRT for a fixed number of samples. After building the tree, it extracts the solution from the tree for execution. If the planner did not find a solution, as is a possible outcome with a probabilistic planner, it returns the best safe alternative.

```
HRT_RRT_JOB:
  update map
  q_init = current robot state
  q_goal = current goal state
  T = BUILD_RRT(q_init, q_goal, FIXED_SAMPLE_COUNT)
  publish EXTRACT_PATH(T)
```

Figure: Pseudo-code for the Hard-real-time RRT planner.

In his experiments in empty, infeasible, and partially-obstructed workspaces, Walker shows that his WCET analysis correctly found the upper limit of the execution time of the planning jobs. There is a wide variation in the job's execution times in the empty and partially-obstructed workspaces, since RRT stops as soon as it finds a solution. In the infeasible workspace, that is, one in which there is no solution, RRT never terminates early with a solution, and thus shows a much narrower variation in execution times that approaches the WCET bound.

In his results, Walker also shows the shortcoming of the sample/time-bounded RRT--that is the probabilistic nature of the planner means that it does not always find a solution. As the planner adds more samples, it probability of finding a solution increases--however probability 1.0 is only reached asymptotically. Even in the empty workspace experiment, in which a solution is a straight-line from

initial to goal configuration, the randomized sampling strategy of RRT does not find a solution in all runs. This provides strong motivation to have the planner produce safe alternative plans when it does not find a solution.

Real-time Motion Planning in the DARPA Urban Challenge

In the DARPA Urban Challenge, responsive motion planning performance was essential to competing successfully. However, because these were essentially research projects, little attention seemed to be paid to hard-real-time system verification processes. In the Urban Challenge, to complete successfully, the planner needed produced a collision-free motion towards the goal sufficiently frequently. In a future consumer market in which millions of vehicles' motions will be computed, the planner must be certified correct. In this section we describe three finalist in the DARPA Urban Challenge, and their approach to real-time motion planning.

The DARPA Urban Challenge winner, CMU, used a variant of the D* motion planner on a discretized lattice-based action space [Likhachev 2009]. Their planner used heuristics to find an initial, sub-optimal solution quickly. It then used subsequent compute cycles to optimize the solution or to incorporate new data into their plan, with no stated bound on compute time. In an example scenario, their planner produced a solution in about 100 ms, and fully optimized it by 650 ms. In [Likhachev 2008], they also present the effect of their heuristic on solution time, showing that a solution found in 60 ms using one heuristic is found in 3490 ms using different heuristic. This suggests that the discretization and search algorithm they used could potentially have a much higher WCET. While a task with a low average compute time and a high WCET is problematic for a building a hard real-time task, they propose to account for this problem by using pre-compute time whenever possible.

Coming in second place, Stanford Racing used a motion planner described as a hybrid A* search on a discretized grid of 160 m x 160 m x 360° at a resolution of 1 m x 1 m x 5° [Dolgov 2010].

At this discretization, there are 1.8 million grid cells to search through, which implies that with a sufficiently fast processor a WCET could be quick enough to be schedulable and responsive. The coarse resolution of their grid produced "unnatural swerves", for which they compensated by using a post-processing smoothing step. The net result was that their planner computed motion plans in under 300 ms, though no rigorous analysis was performed to find the actual WCET.

MIT's fourth place entry used a time-bounded RRT [Kuwata 2009], of a similar form to the hard-real-time RRT planner described in the previous section. The MIT planner ran at 10 Hz, and computed 700 samples per second. Instead of running for a fixed number of samples, as in Walker's planner, the MIT planner stopped sampling based upon polling the compute time. In the event that the planner could not find a solution, it attempted to find a best safe alternative. Failing that, it would apply the send "E-Stop!" signal, and continue planning next cycle.

Real-time Motion Planning Summary

In this section we described several approaches making motion planning algorithms run in hard-real-time system; described a hard-real-time RRT motion planner with a WCET time analysis; and finally saw application of many of the motion planning approaches in research systems that competed successfully in the DARPA Urban Challenge. Of the methods described, discretized search and random-sampling based approaches both proved successful in completing the Urban Challenge tasks. However, it remains an open question as to which of these methods will be best suited for use in a mass-produced safety-critical consumer market in which hard-real-time constraints will have to be provably maintained.

Automotive Robotic (Experimental) Platforms (Michael J. North)

Building any autonomous system that has multiple (even conflicting) requirements is a complex undertaking. This section describes hardware/software necessities in order to create a platform for experimental robotics envisioned for an autonomous vehicle. The end goal is a platform which is well specified and can be duplicated by other research groups seeking to verify the findings and extend that body of work. The desired outcome is a platform for creating extendable and robust experimental systems that allow experimentation of new robotic concepts for testing and verification purposes.

The very nature of building an autonomous robot that has safety issues inherent to its operation immediately requires the attention of a testing platform with concomitant harnesses, test vectors, and routine analysis of baseline expected conditions. In a small system this can be included as part of the development tasks (Test Driven Development). As the system scales this is going to be much more difficult to contain to a software developer/designer as part of the normal set of tasks; further, as the system grows ever larger an adversarial relationship should exist between designer and tester in order to gain trust that certain goals in design and safety systems are being properly achieved. Initially, this may seem heavy-handed to consider this in the infancy of building such a system, however it pays dividends in the amount of time saved when other parts of the project have transformed into multi-headed hydras and distract the team (possibly as a whole) to track down defects that have gone unnoticed, undetected, and/or unexercised.

Testing Harnesses

Repeated unit and functional testing is the cornerstone of how complex systems are built to working completion. Testing the individual functions/methods for for valid inputs and outputs places a rubber stamp on the the building blocks of the system. Functional testing of the system with custom test

harnesses for testing certain data sets, system under load, system under potentially dangerous (simulated) parameters, etc. get the system to being functional completion.

Even better, publish all the testing parameters for every team member so everyone knows what component(s) of the end game they are likely to be contributing.

Sensors

The set of all possible sensors is a very large universe. Vetting these will take time and expense. The expense is part of the verification process (verifying you have found the right tool for the right job).

- Cameras
- Infrared
- Sonar
- Limit switches
- Shaft encoders
- Accelerometers
- Gyroscopes
- Digital compass
- Strain gauges
- Light sensors
- Thermistors
- Laser Ranging
- Sensor interfaces can be digital, analog, I2C, SPI, CAN
    - Your computer controller must support the interface

Sensor Testing and Calibration

Most sensor vendors will provide sufficient information their data sheets will as to the operate within. The task of applications, watchdogs, and operating system will be to ensure that proper action is taken when the operating ranges of these sensors are violated. The requirements of what to do in case of possible, imminent, eventual, or certain (but unknown as to time frame) will require considerable thought. These types of requirements are the determining factors in how scheduling priorities, causal orderings, and synchronizations will be placed in the various (complex) set of tables the system will eventually be built from.

Operating Systems

To a room filled with operating systems people this is going to sound like heresy -- the choice of which operating system to use is not the fundamental decision. The operating system is has a principal role to play, but the application(s) is(are) and their performance have been the primary focus of anything discussed. The Yocto Project is a decent starting place since this system allows one to create custom embedded distributions rather than fishing around for the best distribution that's out there. Embedded distributions in the past couple of years have a "Flavor of the Week" nature to to them. The people "supporting" these distributions have had a short shelf life as they approach various other milestones in their lives (graduation, job changes, marriage, death, divorce, legal action, bankruptcy, etc.). Yocto is well supported and the argument of being able to invent your own future has always been a strong motivator.

- Player/Stage
  - Good control of sensors/actuators over IP networks
  - http://playerstage.sourceforge.net
- Robot Operating System (ROS)
  - Extensive set of software libraries and tools for Ubuntu and Fedora Linux
  - http://www.ros.org
  - (CAN Bus support!)
- Orocos
  - Extensive kinematics and dynamics libraries
  - http://www.orocos.org/
- Rock Robotics
  - Based on Orocos but simpler to use
  - http://rock-robotics.org/
- JAUS Joint Architecture for Unnamed Systems
  - Originally developed by SAE for DoD Used for AEODRS ordnance disposal robots
  - http://www.openjaus.com/ Ad Hoc Use Linux to front end devices like
- Arduinos
  - Ser2net exports serial interface from Arduino to network connection
  - Leverage Arduino ecosystem for motor controllers and sensors
- LiTMUS -- You guys know all about this one….
  - I have it running on a DJI F550 drone I modified last year
  - Also ran a device monitoring system for a High Altitude Balloon we launched last summer -- most reliable part of the software subsystem.
- QNX

- http://www.qnx.com
- Large swaths of the automotive industry use this. When in Rome you might have to do as the Romans do.

Development Boards

There is no shortage of development board BSP systems for creating anything that has been discussed within the scope of this class. It's conceivable that someone could create their own custom board to suit the final reference platform (see https://www.gumstix.com/geppetto)[3], but given the desire to run computationally intensive computer vision algorithms the nVidia Jetson development board looks like the hands-down winner.

- x86 and ARM boards are easy to find
- BeagleBone
- Minnowboard
- Raspberry Pi **2**
- iCreate 2 (iRobot)
- nVidia Jetson TK1 Embedded Development Board
  - The only board on the market with enough horsepower to object detection algorithms and a terrain mapping application simultaneously

FPGA system boards are a strong possibility for directed imaging applications. They will not be general enough for reading sensor information and coordinating the data management needs of a large application. Any FPGA will need interfacing with the general purpose system (designing an I/O subsystem sounds like nice idea, but do not get upset when you go down that rabbit hole and you might not like what you saw down there!).

COTS vs. Custom Designs

Using Commercial Off-The-Shelf components (COTS) is often the path to the lowest cost, but it is often the lowest common denominator of options. The net effect of a pure COTS implementation would likely be a plethora of intercommunicating COTS boards (that add complexity unto themselves).

[3] Intended to be an example -- the setup price for this is somewhat extreme and there are certainly "better" options

A custom design offers a path to creating a tailored solution, but it also requires having adequate skilled personnel available to perform the necessary design work and subsequent follow through to see the design to final assembly. This is an expensive endeavor, but there are certain devices that would be desirable that bridge gaps between the full custom option and COTS components, To wit, a sensor communications concentrator that collects all the sensor data whereby the applications running on a minimalist redundant system could run reliably and cooperatively to maintain the performance requirements as per scheduling and sensor constraints. Concentrator devices are reasonably straightforward in their purpose, design, and function. They can be thought of as solid state storage arrays. Constructing enhanced memory devices (onboard memory processors) could even offload certain computations from FPGA systems and deliver better overall performance (simulation would be needed to verify such an assertion).

Conclusion

An autonomous vehicle design--like many cyber physical systems--is by its nature a highly multidisciplinary project. The challenges associated with a subset of the overall system, robotics, are themselves non-trivial. Any such vehicle which hopes to interact with the world must be able to perceive, analyze, and react to its surroundings with real-time guarantees. Vehicle localization, task planning, real-time motion planning, and equipment and platform choices are integral pieces of such a system. By examining some relevant approaches to these problems insight can be gained into the types of computation and the equipment that may be needed in a novel and sustainable autonomous vehicle design.

References

[DoD 2008] U.S. Department of Defense. *GPS Standard Positioning Service (SPS) Performance Standard*, *4th Edition*. Washington, D.C. 2008.

[Dolgov 2010] Dolgov, Dmitri, Sebastian Thrun, Michael Montemerlo, and James Diebel. "Path planning for autonomous vehicles in unknown semi-structured environments." *The International Journal of Robotics Research* 29, no. 5 (2010): 485-501.

[FAA 2014] Federal Aviation Administration. *Global Positioning System (GPS) Standard Positioning Service (SPS) Performance Analysis Report*. Washington, D.C. 2014.

[Ferguson 2008] Ferguson, Dave, Thomas M. Howard, and Maxim Likhachev. "Motion planning in urban environments." *Journal of Field Robotics* 25, no. 11-12 (2008): 939-960.

[Howard 2007] Howard, Thomas M., and Alonzo Kelly. "Optimal rough terrain trajectory generation for wheeled mobile robots." *The International Journal of Robotics Research* 26, no. 2 (2007): 141-166.

[Karaman 2011] Karaman, Sertac, and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning." *The International Journal of Robotics Research* 30, no. 7 (2011): 846-894.

[Kavraki 1996] Kavraki, Lydia E., Petr Svestka, J-C. Latombe, and Mark H. Overmars. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces." *Robotics and Automation, IEEE Transactions on* 12, no. 4 (1996): 566-580.

[Khatib 1986] Khatib, Oussama. "Real-time obstacle avoidance for manipulators and mobile robots." *The international journal of robotics research* 5, no. 1 (1986): 90-98.

[Koenig 2002] Koenig, Sven, and Maxim Likhachev. "D* Lite." In *AAAI/IAAI*, pp. 476-483. 2002.

[Kuwata 2009] Kuwata, Yoshiaki, Sertac Karaman, Justin Teo, Emilio Frazzoli, Jonathan P. How, and G. Fiore. "Real-time motion planning with applications to autonomous urban driving." *Control Systems Technology, IEEE Transactions on* 17, no. 5 (2009): 1105-1118.

[LaValle 1998] LaValle, Steven M. "Rapidly-Exploring Random Trees A New Tool for Path Planning." (1998).

[LaValle 2006] LaValle, Steven M. *Path Planning*, Cambridge University Press. 2006.

[Levinson 2007] Levinson, Jesse, Michael Montemerlo, and Sebastian Thrun. "Map-Based Precision Vehicle Localization in Urban Environments." In Robotics: Science and Systems, vol. 4, p. 1. 2007.

[Likhachev 2005] Likhachev, Maxim, David I. Ferguson, Geoffrey J. Gordon, Anthony Stentz, and Sebastian Thrun. "Anytime Dynamic A*: An Anytime, Replanning Algorithm." In *ICAPS*, pp. 262-271. 2005.

[Likhachev 2009] Likhachev, Maxim, and Dave Ferguson. "Planning long dynamically feasible maneuvers for autonomous vehicles." *The International Journal of Robotics Research* 28, no. 8 (2009): 933-945.

[Reif 1979] Reif, John H. "Complexity of the mover's problem and generalizations." In *Proceedings of the 20th Annual IEEE Conference on Foundations of Computer Science*, pp. 421-427. 1979.

[Stentz 1994] Stentz, Anthony. "Optimal and efficient path planning for partially-known environments." In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pp. 3310-3317. IEEE, 1994.

[Urmson 2008] Urmson, Chris, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, M. N. Clark, John Dolan et al. "Autonomous driving in urban environments: Boss and the urban challenge." *Journal of Field Robotics* 25, no. 8 (2008): 425-466.

[Walker 2011] Walker, Andrew "Hard Real-Time Motion Planning for Autonomous Vehicles" PhD diss., Swinburne University, 2011