Applied Formal Methods
Final Project Report: Cars-in-Suits

Ryan A. Decker

---

1. Outline

2. Introduction

Autonomous functionality has become increasingly common in vehicles on the road in the last 5 years. We have gone from seeing cars that can drive themselves only in carefully controlled labs to cars that can functionally drive themselves on general roadways available to consumers worldwide.

One component of the autonomous functionality suite that is become incredibly popular is interstate/highway lane keeping. Seeing all the progress and availability of lane keeping for many consumers led to a moment of frustration as I was driving my 1996 Oldsmobile on highway in Iowa. Long, flat, and straight; I realized that a trained monkey or a child could easily be driving the car while I did, well... anything else. At that moment I thought about the technical aspects of creating a lane keeping system, and realized that it is not actually as difficult as it might seem.

From that point on, I was determined to create an automatic lane-keeping system for my 1996 oldsmobile. I knew that it was possible, but I also realized that I would not be contributing anything to the world by trying to create the best lane-keeping system ever, as there hundreds of engineers with millions of dollars behind them working on that same problem. No, instead my goal was to demonstrate just how simply, easily, and cheap such a system could be implemented.

3. Project Background

This project truly began about a year ago, when I first came up with the idea, and then progress was made by developing the lane detection algorithm shortly after. Quickly following that success I began the very first tests of a full up system, which revealed some serious errors in my software design. Those errors along with some shortcomings with regards to the hardware setup, led to me pausing development for a while. Not long after, the transmission in my oldsmobile blew which led me to take the entire system out while the car was being repaired.

That takes us to the point where this final project enters the picture. I decided that this project would be a great opportunity to further the development of my autonomous lane-keeping. I could make hardware improvements, and then make massive improvements to the software as well. So to summarize, prior to the start of the final project, what I had completed was:

(i) Road line detection algorithm
(ii) Control system software
(iii) Hardware implementation in vehicle
(iv) Hardware control software
(v) Some initial testing revealing faults

4. Work Completed for AeroE Project

As a part of this project I completed the following main tasks:

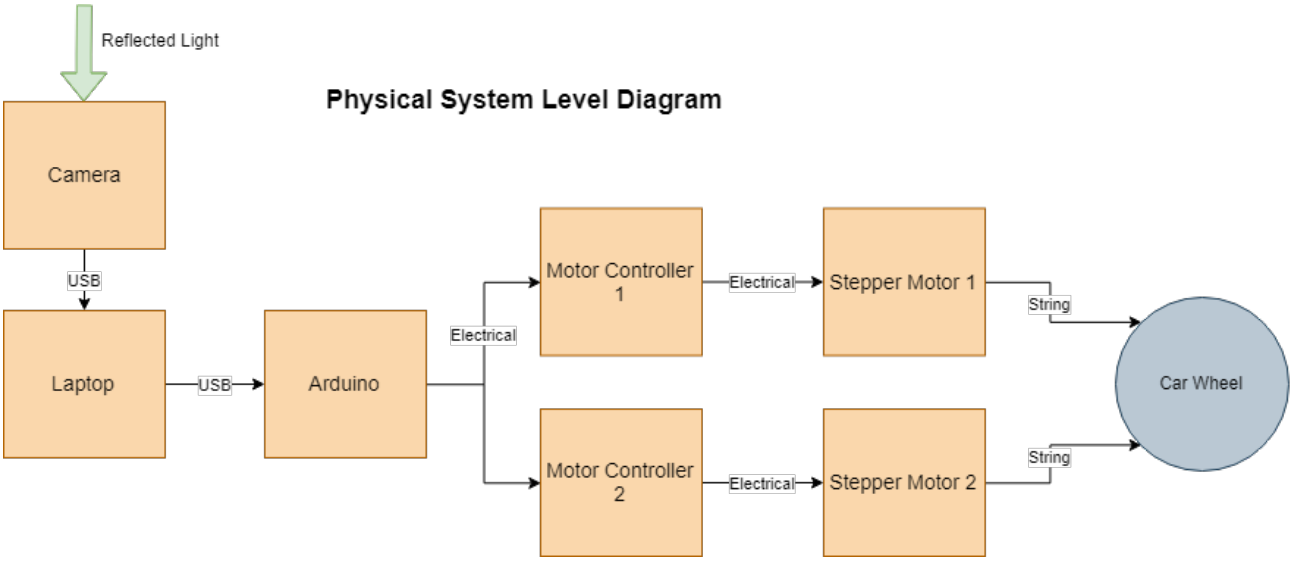
1. Completely new control system
2. Formal verification of control system python implementation
3. Formal proof of safety for control system
4. Minor hardware improvements
5. Further real world testing

Additionally, the tools used in the course of this project are:

1. Keymaera X
2. CBMC
3. Python w/ Pycharm IDE
4. Arduino w/ Arduino C
5. Cython

5. Hardware System

To create the connection between what is on the road, and the actual steering of the wheel I tried to use almost entirely parts that I already had to keep the cost of this project as low as possible. A general systems diagram is shown below.



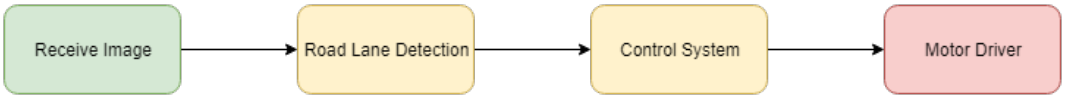
Light reflected by the road is seen by the webcam that I attached to the rear-view mirror of my car. The webcam is connected to the laptop via USB, and the laptop takes each frame from the webcam, performs image analysis on it, and then outputs the control commands to the Arduino. The Arduino interprets those commands and then controls the two stepper motors via their motor controllers. Each of the stepper motors is attached to one side of the wheel via string. To rotate the wheel they always rotate in the same direction, whether left or right.

Some small changes were made to the hardware setup to try and improve the ease of use of the system. First a kill switch was installed on the breadboard to conserve batter power to the stepper motors when not in use, and allow for one more safety measure for when the motors need to be stopped. Additionally a quick release clamp was added to the steering wheel so that the string from the stepper motors is no longer directly attached to the wheel, which makes it much easier to setup and teardown the system for testing.

6. Software System

6.1. Software System Overview

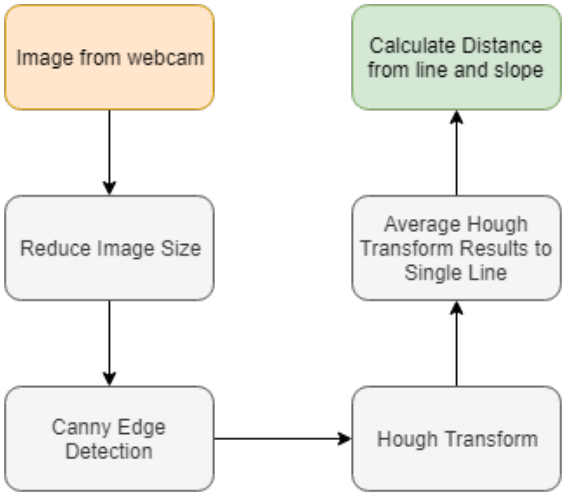
This section is much more interesting and relevant to the this class, as the actual formal verification is used on these systems. A general overview of the software system is in the figure below.



At a high level we must take each image frame from the webcam, find where on the road the lane markings are, and how far away from it the vehicle is which is all performed by the Road Lane Detection Algorithm. Next we must pass that information onto the Control System, which will make the decisions about what state we need to be in to maintain heading. The Control System must then send that information over serial to the Arduino which is running the Stepper Motor Driver, interpreting the Control Systems command and converting it into actual rotation of the motor.

6.2. Road Lane Detection Algorithm

The Road Lane Detection Algorithm is really the most fundamental part of this entire project. If we were not able to detect the road markings, then we would absolutely not be able to advance any further. A block diagram detailing the process that each image frame goes through to extract the location of the road line is below. Additionally, here is a link to see a video of it in action: **https://youtu.be/exCSt00vosI**

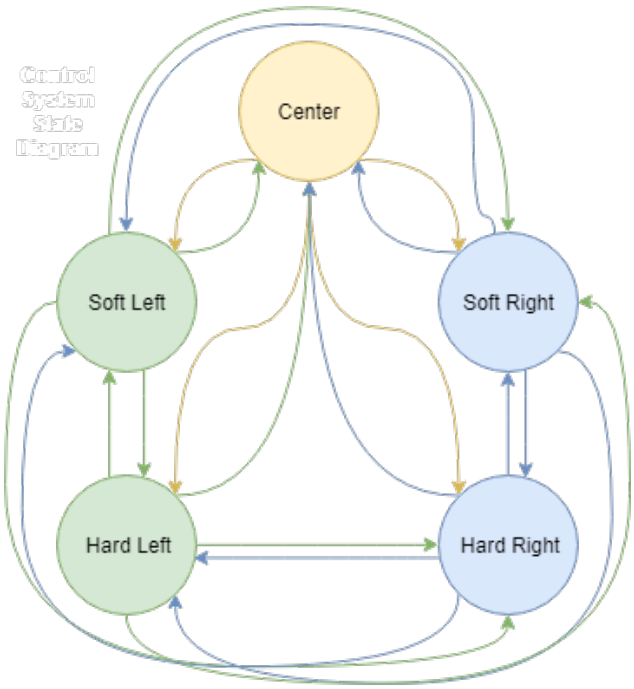


From the above diagram, it may not be entirely clear exactly what is happening and why. First the image frame has it's sized reduced so that we can process it much faster. Then using the OpenCV library in

Python, Canny Edge detection is performed on the image which reveals all of the edges in the image. Next the Hough Transform is used on the image resulting from the edge detection. This transform will take all of the detected edges in the image, and convert them into lines that span the entire image length. These lines represent all of the significant detected edges, and then they are all averaged together to form one single line spanning the whole image that will lie directly on top of the lane markings. From this single line the curvature of the road and the vehicles distance to the lane marking can be extracted.

6.3. Control System

The control system is the decision making component of the system. Given the information from the lane detection algorithm, it will tell the motors to turn clockwise or counterclockwise, corresponding to one of several different states. The first control system I created before I began this revision project and created an entirely new control system had way too many possible states, 23,040 to be exact. Additionally somewhere along the way I implemented the control system incorrectly, and there were bugs in the code that I was unable to find before deciding that an entirely new control system was appropriate. The new control system is shown below.



This control system is both an improvement on the previous control system, and also a vast simplification. This control system contains only 5 states, 2 for going left, 2 for going right, and 1 to stay center. You can go to any state from any other state, which allows for rapid transitions.

7. Formal Verification

7.1. Model Checking Control System Implementation

Before anything else, I wanted to verify that the control system that I had programmed in Python was actually indeed the control system that I had drawn in the block diagram. To do this I tried to find programs that would natively check Python code with my own custom safety/liveness statements. However, that does not exist as far as I can tell. Thus, I decided to port the control system over from Python to C and then use Divine 4 to check the implementation. After using Cython to convert the Python to C code, I could immediately tell that it was not going to be usable to check due to its incredibly messy nature. At that point I decided that the best course of action was to manually convert the Python to C. Luckily the control system was fairly easy to capture in C, and it was easy to verify that the C version was identical to the Python version just via inspection. Next when trying to use Divine 4, I was unable to actually get the tool working on my machine, so I instead switched tools again to using CBMC. CBMC worked perfectly for what I was trying to do and allowed me to enter my own assertions to check all the safet conditions of my system. To do that I actually just checked all of the transistions to make sure each transition was actually possible, because each state is reachable from any other state.

The result of that was successful verification of my control system implementation! The assertions that I checked were all of the form:
$\square(state_i \rightarrow \chi state_j)$ where i, j are any state [0-4]

7.2. Theorem Proving the Control System

The bigger piece of this project, and the aspect that was by far the most challenging was actually proving that the control system that I had designed would guarentee that we would stay inside of the lane using a theorem prover. I opted to use Keymaera X and differential dynamic logic to solve this problem because it would allow me to seemlessly describe both the physical motion of the vehicle along with it's control system decisions.

First I modeled the differential equations that describe the system I want to prove, which are shown below.

Variables:

i) $\theta :=$ Angle of wheel

ii) $slope :=$ Slope of road marking

iii) $V :=$ Velocity of car

iv) $\alpha :=$ Angle of car on road

Constraints:

i) θ maps onto α

ii) $-10° \leq \theta \leq 10°$

iii) V is constant

System Equations:

i) $dist(t) = V \cdot t \cdot sin(\alpha) + dist(t_0) + slope(\tau) \cdot t$

ii) $dist'(t) = V \cdot sin(\alpha) + slope(\tau)$

This systems of equations was then encoded into a Keymaera X model, shown below. Keymaera X has a very steep learning curve, especially when only using material available freely online, and creating this model was a big challenge for me, and it took me a lot of time to gain the understanding necessary to create it.

```
Definitions
    Real V = 30;  /* Velocity of car */
    Real LA = 10; /* Look ahead distance */
End.

ProgramVariables
    Real alpha;   /* Angle of car on road */
    Real dist;    /* Dist from the car to edge of road */
    Real beta;    /* Angle of linear approx. of road line relative to previous frame */
End.

Functions
    Real Sin(R);   /* Declaration of Sin(x) */
End.

Problem
    dist=5.5 & beta=0 & alpha=0 ->   /* Initial Conditions*/
___[
        {
            {?beta>-20 & beta<20; beta:=beta-1; ++ beta:=beta+1; ++ ?beta=-20;
beta:=beta+1; ++ ?beta=20; beta:=beta-1;}            /* Describes how beta can change */
            {?dist<3; alpha:=10; ++ ?dist>7; alpha:=-10;}  /* Simplified Control System */
            {dist'=Sin(alpha)*V+Sin(beta)*LA}        /* Differential Equation of motion */
        }*      /* Repeat nondeterministically */
___]dist>=1 & dist<=9    /* Safety Condition */
End.
```

However this model has a problem. The problem is of course that I am too inexperienced with theorem proving and Keymaera X to actually prove it. I was unable to prove this model, which describes the entire system, so instead I went and reduced and simplified the model to only prove a single small part of it, something I was actually capable of (in the time given).

Shown below is the simplified model that I built, and was able to successfully prove! This model simplifies the control system, stating that it only ever can go Soft Left or Soft Right, and also simplifies the road to be straight. This is obviously too simple to be practicle, and only going Soft Left or Soft Right would cause the car to bounce around the lane horribly, but it is a nice step, and gives me confidence that with more time, I would be able to successfully prove the entire control system.

```
Definitions
  Real V = 30;  /* Velocity of car */
End.

ProgramVariables
  Real hvel;
  Real dist;
End.

Problem
  dist=8 & hvel=0
  ->
  [
                {
                        {?dist<3; hvel:=1; ++ ?dist>10; hvel:=-1;}
                        {dist'= hvel & dist>=0}  /* Differential Equation of motion */
    }*@invariant(dist>=0)
  ] dist>=0
End.
```

And of course, here is the proof of successful proof.

The above image shows that the tactic "master" was used to solve the proof, which means Keymaera can auto prove the theorem, and that my safety condition is never violated (i.e. that the car never drives over the right lane).

8. Summary

I was successfully able to implement an entirely new control system for the vehicle, then I was able to formally verify that the implementation was correct using CBMC, and finally prove that a weaker version of the control system will never result in it crossing over the right lane line. Additionally I was able to to some more testing on actual roads, that demonstrated that the system is now software ready, but simply needs more hardware improvements before truly operational.

9. References

1. C. Y. Low, H. Zamzuri, S. A. Mazlan, "Simple robust road lane detection algorithm", 2014 5th International Conference on Intelligent and Advanced Systems (ICIAS), pp. 1-4, 2014.
2. "Mitsch, Stefan, et al. On Provably Safe Obstacle Avoidance for Autonomous Robotic Ground Vehicles. Symbolaris, 2012, symbolaris.com/pub/robix.pdf."
3. Platzer, Andre. KeYmaera X: An AXiomatic Tactical Theorem Prover for Hybrid Systems. KeYmaera X: An AXiomatic Tactical Theorem Prover for Hybrid Systems, 1 Oct. 2018, www.ls.cs.cmu.edu/KeYmaeraX/.