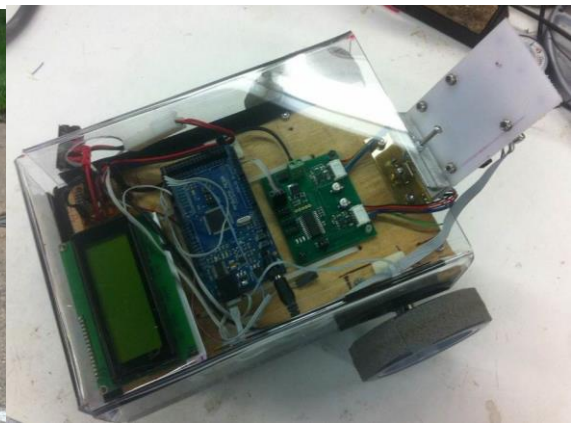
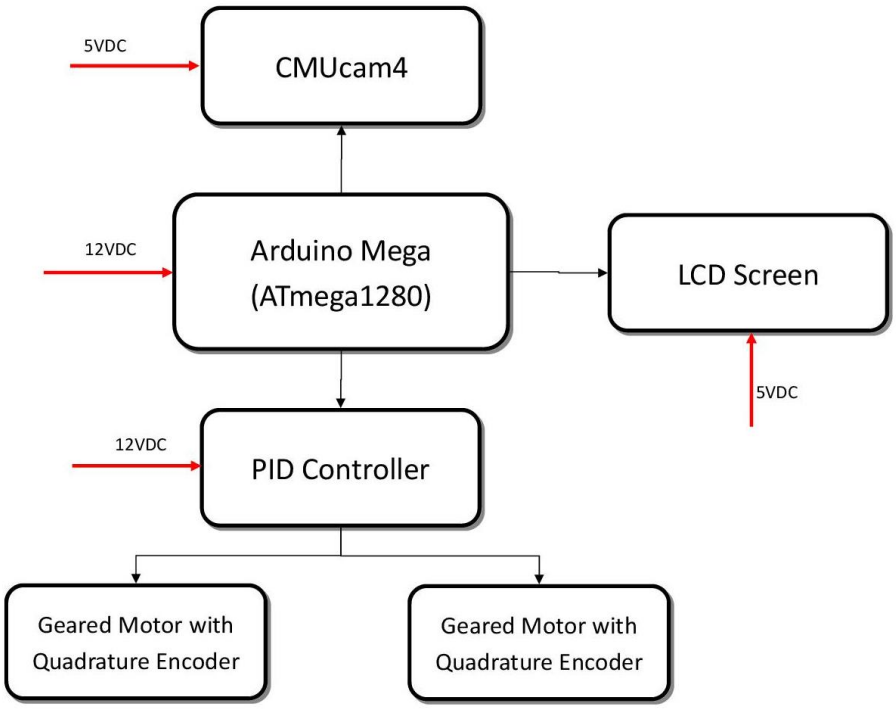
Moboss: Entry for 2013 Mobot Competition
Team: Cosku Acay, Edwin Cho, Kenneth Li, Nishant Pol, Oguz Ulgen, Sam Zeng

Project Description
In Carnegie Mellon's Mobot competition, robots must pass through a series of 14 gates positioned over a 255-foot curved white line on an outdoor sidewalk. The course includes two steep ramps and a decision point area where the robot must select the correct route to pass through 14 gates in serial order. Since the line is outdoors and under Pittsburgh's weather, the robot must function in a variety of conditions. This year it was rainy and windy on competition day. More information can be found at http://www.cs.cmu.edu/mobot/mobot.html



Solution with the CMUcam4
We chose to approach the line-following task using the CMUcam4 because it required minimal hardware and had an interface well suited for tracking a line.

Hardware Description



The CMUcam4 captures an image of the line at 30 frames per second and feeds the color tracking image to the Arduino Mega via one of the three Serial lines. The Arduino Mega then processes this image (please see Software Description below) and delivers motor speed and turn commands to the PID controller via another Serial line. The PID controller in turn provides the Arduino Mega with encoder counts for both right and left wheels, which allows us to keep track of our position and heading relative to gate 0.

The PID controller (MD-25) is made by Robot-Electronics in the UK. It drives two geared DC brush motors with motor shaft magnetic encoders. The LCD screen is for debugging purposes. We had two power rails, 12V for motors and 5V for logic, sharing the same ground and both sourced from a single 12V rechargeable battery.

We initially planned to make a prototype from a wood base to get us into the programming phase quickly, then work on a faster, more reliable base such as an RC car. However, we found that the prototype was very reliable and easy to work with, so we kept it for the final design. Waterproofing for the body consisted of a layer of cellophane wrap under a layer of packaging tape. Rain however leaked through the packaging tape but fortunately not through the cellophane wrap. The camera was covered by a square of plastic from a Ziploc bag since we found that the "filter" improved the camera's ability to ignore small patches of the line that, to the camera, looked like the background concrete.

Algorithm Description

Background:
We use the CMUcam4 in color tracking mode with YUV mode enabled, which gives us a binary image representation of the line. The camera software filters out anything that is not white (i.e. the background) and puts an "on" pixel where there is a line (where it is white). We use YUV mode so that color tracking is insensitive to the intensity of light, so that our robot can function under a wide variety of conditions including indoors and outdoors.

Line Following:
The CMUcam4 provides our Arduino chip with an 80 by 60 binary image. We sample 10 equally spaced rows of pixels from the image and calculate the centroid of "on" pixels (see here: https://en.wikipedia.org/wiki/Centroid#Of_a_finite_set_of_points). We then assume that the line lies on the centroid. By doing this on 10 different rows in the image, we get a good approximation of the line by connecting the dots. We can then use this data by finding how far each point on the line is from the vertical line that lies at the center of the image to get a value representing how curved the line is. This can be seen as something like finding the area between the tangent line and a curve and using that area to determine how much we need to turn. The sign determines if the turn is left or right, while the magnitude determines how much we need to turn to follow the line. As you would expect, if we measured zero area, then the line is directly in front of the robot and is a straight line. If the magnitude of the area is large, then the robot needs to turn to maintain attempt to minimize the magnitude of this area.

Decision Points:
To determine if the robot is at a decision point, we take the data the camera gathers from line following and check to see how many average center points appear in black space. This works based on the assumption that the average point between two lines should not be on either line, unless the lines are the same line. If the robot detects multiple average points in the black space, it can conclude that there must be two lines here. Then it selects the appropriate turn and biases its turning toward that direction until it sees only one line.

Please go to https://github.com/cacay/Mobot to see a simplified version of our code.