McMaster University

Inspiring Innovation and Discovery

Autonomous Vehicle Control System

Verification and Validation

(Deliverable #10)

GROUP #1:

Colin Lobo

Mohit Bhagotra

Steven Back

Gregory Mainse

Hector Valdez

Matthew Dawson

April 01, 2012

Revision 3.0

Revision History

Revision Number	Author(s)	Description	Date (D/M/Y)
1.0	Steven Back	Creation of Document and insertion of content	23/02/12
1.1	Mohit Bhagotra Greg Mainse	Modified serial communication modules to reflect current changes and state.	27/03/12
2.0	Steven Back	Added Tractor pull, camera module – lane following, camera module – obstacle detection modules	30/03/12
2.1	Hector Valdez and Steven Back	Added lane detection images to document	30/03/12
2.1	Hector Valdez	Added function tables to path choice and added the path correction module	30/03/12
2.2	Greg Mainse	Completed V&V for the new Beagleboard serial module	31/03/12
2.3	Colin Lobo	Added Tractor Pull Module test case data	31/03/12
2.4	Mohit Bhagotra	Completed Serial Transmission V&V from the Arduino side. Also completed Arduino Timing tables with updated design configuration.	01/04/12
2.5	Steven Back	Formatting and Matt’s obstacle detection	01/04/12
3.0	Colin Lobo	Proofread document for final submission	01/04/12

Table of Contents

1. INTRODUCTION

1.1 Document Purpose

The purpose of this document is to effectively communicate the verification and validation of a system which meets the requirements stated in the SRS. This includes the testing and simulation of the hardware and software components. The reader should by the end of this document have a clear understanding of:

1. How the hardware components behave
2. The testing of the Arduino and Beagleboard components/modules
3. The testing of timing constraints that is present with the system design
4. Potential issues that arose during testing

1.2 System Scope

1.2.1 Purpose

The purpose of the project is to provide the client a well-documented and complete engineering system that completes the following tasks:

1. A system that will operate autonomously
2. A system that will follow and stay within a lane of a predefined track (REF SRS 2.4) at all times
3. A system that will avoid inanimate obstacles in its path
4. The system's final design will cost no more than $750.00

1.2.2 Goals

Found below are the goals for this project in no particular order. Further explanations of each individual goal can be found in the SRS (REF SRS 1.2.2).

	GOAL (not in any particular order)
1	KISS – Keep it Simple Stupid
2	Don't crash
3	Stay within the boundaries of road
4	Stay completely within one lane when not changing lane
5	Drive forward when not avoiding obstacles
6	Emergency stop
7	Go as fast as possible without issues
8	Smooth motion of vehicle
9	Advertisement opportunities- money donated if profit
10	Follow other moving object
11	Look good
12	Stay within budget
13	Cost effective
14	Good code

Table 1: List of goals for project

1.2.3 Project Scope

The scope of this project is to test the ability of students in the Mechatronics and Software Engineering disciples to culminate all that they have learned in their academic careers and apply it to a major year-long project. The development team will be designing a system (1/10 scale car) which can autonomously navigate and avoid obstacles in a closed environment provided by the client. Any functionality other than described in the SRS is considered out of the project scope.

1.3 Intended Audience and Document Overview

The intended client of this document shall be Dr. Alan Wassyng, while the audience also comprises of Dr. Wassyng's teaching assistants and the project developers. The document is thus tailored towards a technical, well versed, audience that is familiar with basic technical terminology. The document is also tailored towards the developers of the project and is thus organized in a very methodical structure to allow for the design and final product to reference the DSD with ease and create a seamless transition between documents.

1.4 Document Conventions

1.4.1 Naming Conventions

The following naming conventions are observed in this document:

k_ : constant value
m_: monitored variable
c_ : controlled variable
e_ : enumerated values
y_ : enumeration
i_ : input variable (individual component)
o_: output variable (individual component)
d_: data variable (data in communications packet)

The first letter of the constant shall be lower case, and all subsequent starting characters are upper case:

Ex. k_MyDogSkip

Previous values shall be represented by a subscript "-x" where x represents how far in the past

Ex. k_MyDogSkip_{-2}

1.4.2 Formatting Conventions

- **Paper size:** US letter (8.5"x11")
- **Margins:** top margin: 0.6", bottom margin: 0.5 ", inner margin: 0.75", outer margin: 0.75", header and footer 0.3" from edge.
- **Header:** Each page shall have a header with the following attributes:
  - ➢ Font and size Times (New) Roman, Bold, 14 point for portrait oriented documents
  - ➢ Font and size Times (New) Roman, Bold, 18 point for landscape oriented documents (e.g. PowerPoint)
  - ➢ Line below, with 2 points separation from text
  - ➢ Left, aligned with margin: the month and year of the publication (the venue date)
  - ➢ Right aligned to the margin: the document designator, which includes the document number:
  - ➢ doc.: Autonomous_Vehicle_Control_System_XX_RevY
  - ➢ Where:

➢ XX is the abbreviation for the document (SRS – System Requirements Specification)
➢ Y is the revision number

- **Footer:** Each page shall have a footer with the following attributes:
  - ➢ Font and size Times (New) Roman, Normal, 12 point
  - ➢ Line above
  - ➢ Left, aligned with margin: the word "Submission"
  - ➢ Center: the word "page" followed by the page number
  - ➢ Right aligned to the margin: the first author and company (in the format: author_name, company).
- Every document submission must have an author and company as a point of reference for the submission.

1.5 References and Acknowledgements

- The formatting guidelines have been adapted from the IEEE 802.22 Documents guideline

Example: Internet URL:
Author(s)*. “Title.” Internet: complete URL, date updated* [date accessed].
M. Duncan. “Engineering Concepts on Ice. Internet: www.iceengg.edu/staff.html, Oct. 25, 2000 [Nov. 29, 2003].

2. Arduino Module Testing

2.1 Obstacle Sensor Data Module

Updates: The current design uses a PlayStation Eye instead of the set up described below. This solves most if not all potential problems described below.

Motivation: This is a key component of the system. One of the main requirements of the system is to be able to detect an obstacle in all lanes at all times. To verify we are fulfilling this requirement this test is necessary. If successful we can be sure that the system will be able to detect an obstacle in all scenarios. (Requirement: SRS 3.3.13)

Procedure:
The vehicle will have the sensor configuration as follows the all the following tests as determined from hardware testing:

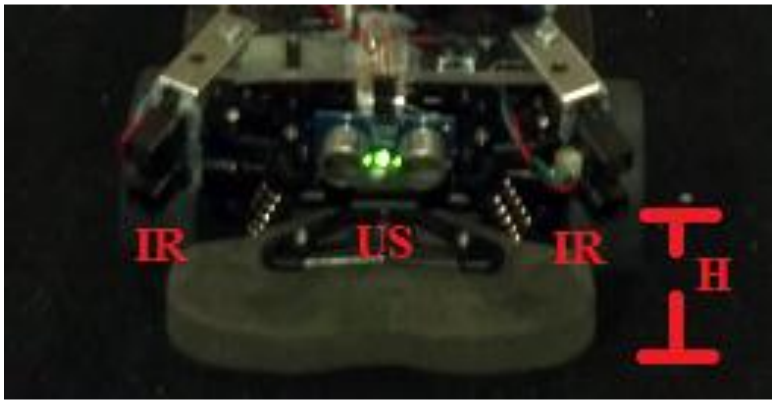


Figure 1: Front bumper view with sensor configuration. Sensors are elevated H=7cm height from ground. US – Ultrasonic Sensor, IR – Infrared Sensor

Test #1: The vehicle will be placed in the middle of the centre lane, 1 metre away from all three obstacles as pictured below. This test will also take place on a straight part of the track. The vehicle will then be placed closer to the obstacles and readings from all three sensors will be recorded. The Ultrasonic will be outputting a distance reading in cm which will be checked against the actual distance using a measuring tape. The two IR sensors, positioned at an angle of 20 degrees from center, will be outputting a boolean value, 1 if obstacle is detected and 0 if no obstacle is detected. The module is set up in a way such that 1 is outputted whenever the following occurs, reading < min_distance. Note: min_distance for this test will be 45 cm.

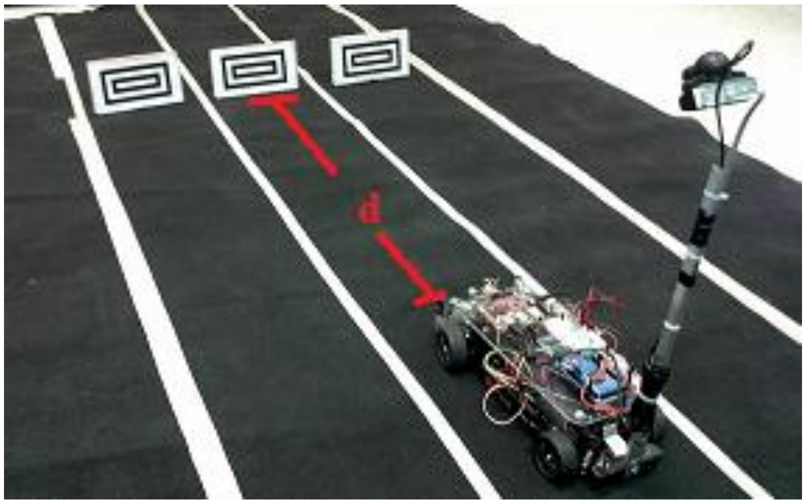


Figure 2: Test #1 set up. When d =< 45 cm the IR's should output.

Distance d (cm)	US Expected Reading (cm)	US Actual Reading (cm)	Right IR Expected Reading (bool)	Right IR Actual Reading (bool)	Left IR Expected Reading (bool)	Left IR Actual Reading (bool)	Test Result
100	100	100	0	0	0	0	Pass
80	80	82	0	0	0	0	Pass
60	60	61	0	0	0	0	Pass
50	50	51	0	0	0	0	Pass
45	45	44	0	0	0	0	Pass
40	40	40	1	1	1	1	Pass
35	35	34	1	1	1	1	Pass
30	30	28	1	1	1	1	Pass
20	20	20	1	0	1	1	Fail
10	10	11	1	0	1	0	Fail
0	4	4	1	1	1	1	Pass

Potential Problems:

1. The vehicle must be in the middle of the lane for the IR sensors to detect objects in the adjacent lanes otherwise the system risks missing an obstacle. Potential solution is to adjust the IR angle for far left and far right conditions. So regardless of the position of the vehicle within a lane an obstacle can still be detected in an adjacent lane.
2. From this test we can see we will not be able to fulfill our requirement of detecting obstacles in all lanes at all times when we are in the inside or outside lanes nor on any curves with our current set up.

Test #2: Same as Test #1 but the module is set up in a way such that 1 is outputted whenever the following occurs, reading < min_distance. Note: min_distance for this test will be 70 cm. The center obstacle will be 1m away from the front of the vehicle while adjacent obstacles will be 130cm. This causes a scenario in which the car should come to a halt since there is no room available to switch back to the center lane.

Distance d (cm)	US Expected Reading (cm)	US Actual Reading (cm)	Right IR Expected Reading (bool)	Right IR Actual Reading (bool)	Left IR Expected Reading (bool)	Left IR Actual Reading (bool)	Test Result
100	100	99	0	0	0	0	Pass
80	80	83	0	0	0	0	Pass
60	60	61	1	0	1	0	Fail
50	50	50	1	0	1	0	Fail
45	45	45	1	1	1	0	Fail
40	40	39	1	1	1	0	Fail
35	35	34	1	1	1	0	Fail
30	30	33	1	1	1	1	Pass
20	20	21	1	1	1	1	Pass
10	10	11	1	1	1	1	Pass
0	4	5	1	1	1	1	Pass

Potential Problems: There is a blind spot when we are far away from an obstacle. Since the IR beam is narrow there is a potential to miss the adjacent obstacles. Solution to this may be to have 2 more IR sensors at slightly offset angles than the original two to cover a much larger area.

2.2 Speed Encoder Module

Motivation: One of the key system requirements is to be able to travel as fast as possible around the track. This requires control of the vehicles speed (ie. fast on straightaways and slower on curves to maintain control). The speed encoder module allows the system to monitor the current speed of the car at all times. Verifying this module is functioning correctly will aid in meeting the above requirements. (Requirement: SRS 3.4.1)

Procedure:

The Arduino MEGA 2560 will provide a print out of the module’s returned value in RPM. We will manually spin the wheel at a fixed rate for a 30 seconds and verify the output by reading the print out.

Number of turns in 30 seconds	Expected RPM (# of turns in 30s * 2)	Serial Print Out (RPM)	Test Result
0	0	0	Pass
1	2	2	Pass
5	10	10	Pass
10	20	20	Pass
30	60	60	Pass

Potential Problem: There is no way with our current setup to confirm if the output RPM from the module at very high RPM rates is accurate.

2.3 Tractor Pull

Motivation: One of system requirements is to be able to complete a tractor pull as our vehicles unique task. (Requirement: SRS 3.3.18)

Procedure:

The vehicle will pull behind it a cart in which 2.5 pound weights can be placed up to a maximum of 20 lbs. Each lap the vehicle will have an additional 5 pounds placed into the cart and the lap time be measured to verify the speed controller is dynamically adjusting the vehicles speed to stay constant.

Weight (pounds)	Lap Time (sec)	Test Result
0	20.6	Pass
5	19.9	Pass
10	21.8	Pass
15	25.1	Fail
20	27	Fail

Potential Problem: The available torque from the motor limits the vehicle to a maximum of 10 lbs before the performance decreases due to physical limitations of the motor.

2.4 Actuator Module

Motivation: Verifying that the modules which control the motion of the vehicle are functioning correctly is critical. In turn, if these tests are successful, it validates that we are meeting the system requirements. Requirements such as, the vehicle shall be able to stay within one lane when travelling (Requirements: SRS 3.3.3), the vehicle shall have the ability to stop (Requirements: SRS 3.3.15), shall travel with smooth motion (Requirements: SRS 3.4.2), travel as fast as possible around the track and maintain full control at all times when travelling (Requirements: SRS 3.4.1).

Servo: [http://www3.towerhobbies.com/cgi-bin/wti0001p?&I=LXUK84]
[http://www.societyofrobots.com/actuators_servos.shtml]

Motor: [http://www3.towerhobbies.com/cgi-bin/wti0001p?&I=LXAXVD&P=7]
[http://www.societyofrobots.com/actuators_dcmotors.shtml]

Procedure:

Test #1: Verifying we have full control over the steering using the built in servo library provided by the Arduino IDE. We can simply write a specified angle (0 – 180) and record the wheel angle using a protractor.

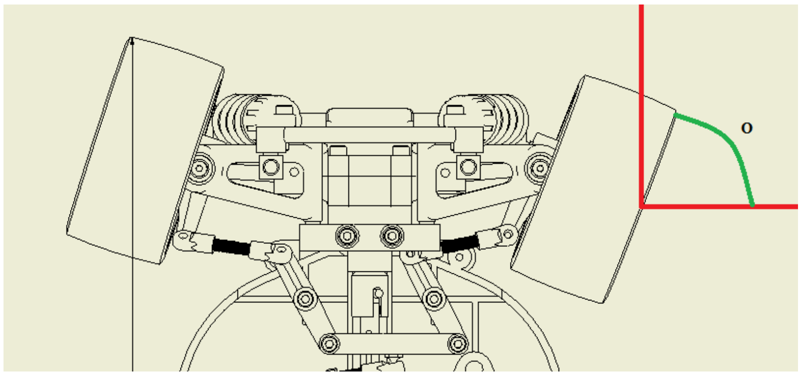


Figure 4: Servo test set up. Angle, O, to measure.

servo.write(x)	Expected wheel angle (degrees)	Actual wheel angle (degrees)	Test Result
0	30	50	Fail
10	30	50	Fail
20	30	50	Fail
30	30	50	Fail
40	40	50	Fail
50	50	50	Pass
60	60	60	Pass
70	70	70	Pass
80	80	80	Pass
90	90	90	Pass
100	100	100	Pass
110	110	110	Pass
120	120	120	Pass
130	130	130	Pass
140	140	130	Fail
150	150	130	Fail
160	150	130	Fail
170	150	130	Fail
180	150	130	Fail

Potential problem: Our original requirements required a steering window of 120 degrees (60 degrees either side of the axis of the center of the car), which form our expected values. This test proves the servomotor is limited due to its physical configuration and therefore instead of 120 degrees the actual window is 80 degrees. We have modified our requirements to reflect this finding. The good news is that this does not impact the performance of the system as it is able to go through the track comfortably and stay within the lanes at curves.

2.5 Serial Transmission Module

Updates: The design has been modified eliminating the communication from the Arduino to the BeagleBoard. This is due to the fact that the whole obstacle detection system has been shifted from specialized sensor hardware to the camera, which is interfaced with the Beagleboard. Removing this module has eliminated the previous errors we encountered when the Arduino was communicating with the BeagleBoard. It has also sped up the system as we no longer have to poll and external sensors which was the most time consuming part of the whole process. Hence we are going to test to see if the Arduino correctly receives packets from the Beagleboard and if there are any problems occurring on the actual serial transmission line.

Motivation: To round out the system a correct serial transmission protocol is required to communicate from the Arduino to the Beagleboard. Without the correctness of this module we fail to meet many of our requirements as the results of the systems actions depend on the decisions made by both boards whose information complement one another. (Requirements: 3.4.3)

Procedure: Test packets will be sent over the serial line and a pair XBEE radios will be used to verify the integrity and validity of the data transmitted back and forth between both boards. The packets are composed of the following depending on the board being transmitted from:

Beagleboard → Arduino: [Header packet] + [Motor speed] + [Steering angle]

Packet sent	Expected packet received	Actual packet received	Test result
[105,90]	[105,90]	[105,90]	PASS
[105,50]	[105,50]	[105,50]	PASS
[105,130]	[105,130]	[105,130]	PASS
[105,70]	[105,70]	[105,70]	PASS
[105,110]	[105,110]	[105,110]	PASS
[101,90]	[101,90]	[101,90]	PASS
[101,50]	[101,50]	[101,50]	PASS
[101,130]	[101,300]	[101,300]	PASS

This series of packets was run approximately 500 times in a continuous loop and the serial transmission system had no problems keeping up with the goal of 30 frames per second for a whole system cycle to transmit information from the Arduino to the Beagleboard. This is due to some modifications from the previous version of the serial transmission code where we have added some robust integrity checks for the buffer space and packet size. Thus the success of this test helps the design meet the requirements of allowing the car to travel around the track as fast as possible.

2.6 Arduino Timing

Motivation: As per the requirements, it is essential that the vehicle has good code and can travel around the track as fast as possible (Requirement: SRS 3.4.1). A key to being able to travel around the track as fast as possible is being aware of the timing limitations within the system and being able to asses if that is sufficient enough to meet the requirement. Secondly, good code also means being able to minimize the critical path.

Procedure: Place all three obstacles in their respective lanes on a straight portion of the track. Output the time to the console that each of the major events (as outlined in the below table) in the Arduino took to accomplish. Repeat for a variety of distances to compare how distance affects timing.

Distance to Middle lane obstacle (cm)	Start Time of main loop (ms)	Time after get Ultra Sonic (ms)	Time after get IR sensor Right (ms)	Time after get IR sensor Left (ms)	Time after tx serial (ms)	Time after rx serial (ms)	Time after angle and speed write (ms)	End of loop time (ms)
∞	12225	12270	12271	12274	12275	12302	12303	12304
100	73658	73688	73691	73707	73708	73736	73737	73738
90	157715	157745	157750	157767	157768	157796	157797	157797
80	198748	198777	198784	198800	198801	198830	198831	198831
70	263987	264016	264023	264039	264040	264068	264069	264069
60	313795	313823	313830	313847	313848	313877	313878	313878
50	357114	357143	357150	357167	357168	357195	357196	357196
40	434019	434045	434055	434071	434072	434100	434101	434101
30	470647	470674	470683	470700	470701	470730	470731	470731
20	510161	510587	510197	510214	510215	510243	510244	510244
10	608544	608570	608580	608596	608597	608626	608627	608627
0	654454	654500	654501	654507	654508	654535	654536	564536

Update: Since revision 1.0 of the V&V document, our designed has been overhauled to conduct obstacle detection from the camera hardware itself without any sensors. As a result columns 2-5 in the above are void while the column “Time after tx serial (ms)” is the new main loop start time. As we can see above, most of the time on the Arduino is spent waiting for the Beagleboard to transmit data, and the transmission itself takes up ~28 ms on average. This allows us to meet our goal of processing at a rate of 30 frames of per second. The below discussion reflects on the bottleneck of timing of polling the sensors from the previous design.

Discussion:

After taking the raw data and putting it in a readable format, we get:

	Ultra Sonic	IR Right	IR Left	Main Loop
∞	45	1	3	49
100	30	3	16	49
90	30	5	17	52
80	29	7	16	52
70	29	7	16	52
60	28	7	17	52
50	29	7	17	52
40	26	10	16	52
30	27	9	17	53
20	426	390	17	53
10	26	10	16	52

It is clear that on average the Arduino takes about 52 ms to complete a full cycle, it won't be known until testing on the track is done to see if this meets the requirement of being able to go around the track sufficiently fast.

3. Hardware Testing

Update: The current set up does not include the use of the IR sensors or the Ultrasonic sensors described in the following sections. These sections have been included however to document our design decisions and for future reference.

3.1 IR Sensors

(SHARP GP2Y0A02 [20-150cm range])

Datasheet: http://www.sharpsma.com/webfm_send/1487

Motivation:

Test #1: Test one is used to verify that the sensor is functioning correctly, by collecting and interpreting sensor data, which is critical. It also aids in understanding how to most effectively use the sensor and set it up (ie. distance or trip sensor, signal conditioning, orientation etc.).

Test #2: Test two is used to create a simple LUT or representative distance equation (reading → cm).

Test #3: Test three is used to identify the optimal angle, O, to detect an obstacle in an adjacent lane at distance, d. This is in order to fulfill requirement of detecting obstacles in all three lanes at any time (3.3.13).

Procedure:

Test #1: An obstacle, as described in the requirements, is placed at fixed positions away from the IR sensor. The IR sensor will be orientated vertically as described in the datasheet. A 10uF capacitor will also be placed between Vcc and Ground in order to stabilize the power supply line as recommended in the datasheet. An oscilloscope will be used to measure its output voltage and compared to that of the expected output voltage from the sensor's datasheet.

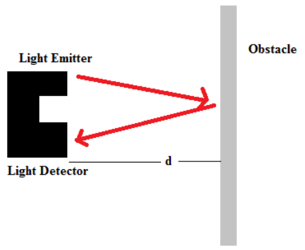


Figure 1: Test #1 set-up (side view)

Distance (cm) [d]	Expected Output (V)	Actual Output (V)	Test Result (Pass/Fail)
200 (out of range)	<0.45	0.20	Pass
150 (max)	0.45	0.40	Pass
100	0.6	0.68	Pass
80	0.8	0.80	Pass
60	1	1.16	Pass
40	1.5	1.76	Pass
20 (min)	2.5	2.68	Pass
10 (out of range)	<2.5	1.92	Pass

Discussion:

This sensor works well at all ranges although the output voltage fluctuates quite a bit even with a 10uF capacitor. The fluctuation is more prominent at longer ranges (>80cm) sometimes up to +/- 30mV.

Test #2: An obstacle, as described in the requirements, is placed at fixed positions away from the IR sensor. The IR sensor will be orientated vertically as described in the datasheet. A 10uF capacitor will also be placed between Vcc and Ground in order to stabilize the power supply line as recommended in the datasheet. The sensor will also be connected to an Arduino MEGA 2560 analogue pin to create a simple LUT or representative distance equation (reading □ cm). The sensor will take 20 readings (1 reading every second) and the min and max of the 20 readings will be recorded.

Distance (cm) [d]	MIN A/D Output (0-1023)	MAX A/D Output (0-1023)	AVERAGE A/D Output (0-1023) (round-to-nearest int)
200 (out of range)	36	42	39
150 (max)	68	79	74
140	78	86	82
130	86	93	90
120	98	104	101
110	109	113	111
100	126	130	128
90	140	147	144
80	164	168	166
70	193	201	197
60	228	234	231
50	279	286	283

40	343	350	347
30	440	449	445
20 (min)	533	542	538
10 (out of range)	427	435	431

Test #3: An obstacle is placed at O degree angle at d distance. The d distance will only vary from 30 – 90 cm as described in the system requirements (3.3.13). The obstacle will be approximately placed in the center of the right lane. At each distance the angle will sweep from 10 degrees to 90 degrees at 5 degree decrements (Figure 2). The IR sensor will be orientated vertically as described in the datasheet and fixed at the front corner bumper of the vehicle approximately 10cm from the ground. A 10uF capacitor will also be placed between Vcc and Ground in order to stabilize the power supply line as recommended in the datasheet. Since we will be using an Arduino MEGA 2560, we will use the board and record what the sensor's output is via the onboard A/D converter outputs on one of its analogue pins. This is to create a small look up table (LUT) since we only need to detect an obstacle at a certain range (true/false) as required in the system requirements (3.3.13) and not the obstacle's range.

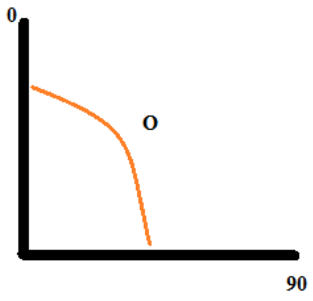


Figure 2: Angle sweep

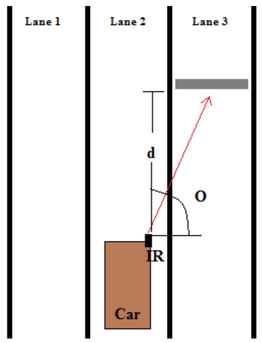


Figure 3: Test #2 set-up

Angle O (degrees)	d = 90 (cm)	d = 85	d = 80	d = 75	d = 70	d = 65	d = 60	d = 55	d = 50	d = 45	d = 40	d = 35	d = 30	d = 25	d= 20
5	128	146	152	160	164	177	206	223	219	-	-	-	-	-	-
10	141	152	148	152	160	181	194	210	219	253	302	-	-	-	-
15	-	128	135	145	156	177	189	220	250	248	265	294	338	-	-
20	-	125	137	148	172	185	185	228	253	245	265	310	363	-	-
25	-	-	-	129	141	164	215	220	236	277	287	295	333	410	-
30	-	-	-	-	-	-	156	186	224	254	270	295	330	380	445
40	-	-	-	-	-	-	-	-	-	-	-	-	-	-	306
45	-	-	-	-	-	-	-	-	-	-	-	-	-	-	112
50	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Note: A dash (-) represents no detection.

Discussion:

As we can see to maximize our range for obstacle detection in an adjacent lane we should have the sensor angled at approximately 20 degrees. If we need to detect obstacles closer to the vehicle we should have the angle greater (>20 deg) and if we like to detect obstacles at farther ranges we need the angle less (<20 deg)

3.2 Ultrasonic Sensor

((PARALLAX PING) US Distance Sensor [2-300cm range])

Datasheet: http://www.parallax.com/Portals/0/Downloads/docs/prod/acc/28015-PING-v1.6.pdf

Motivation:

Test #1 is used to verify that the sensor is functioning correctly, by collecting and interpreting sensor data, which is critical. It also aids in understanding how to most effectively use the sensor and set it up (ie. distance or trip sensor, signal conditioning, orientation etc.).

Procedure:

Test #1: An obstacle, as described in the requirements, is placed at fixed distances perpendicular to the US sensor. The US sensor will be horizontal as described in the datasheet. The sensor will also be connected to an Arduino MEGA 2560 digital pin with the example sketch, PING provided by David A. Mellis in the public domain which converts the sensors readings to centimeters.

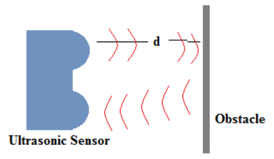


Figure 4: Test #1 set up

Distance (cm) [d]	Expected Output (cm)	Actual Output (cm)	Test Result (Pass/Fail)
320 (out of range)	??	371	Pass
300 (max)	300	310	Pass
250	250	254	Pass
200	200	202	Pass
150	150	151	Pass
100	100	101	Pass
80	80	81	Pass
60	60	63	Pass
40	40	43	Pass
20	20	22	Pass
10	10	13	Pass
5	5	8	Pass
2 (min)	2	3	Pass
1 (out of range)	??	4	Pass

Note: ?? represents unpredictable value

Discussion: Since the sensor is already placed behind the bumper, by approximately 2 cm, we will never have an obstacle in a region less than 2 cm, which is convenient since the sensors output is unpredictable in this range. Another important thing to note is that the obstacle must be completely vertical at longer ranges (>80cm) in order to maximize obstacle detection, otherwise the reading is maximum output (~371) even with a slant as little as 10 degrees.

3.3 Hall Effect Sensor

(HAMLIN 55140 Flange Mount Hall Effect Sensor)

Datasheet: http://www.hamlin.com/specsheets/55140%20IssueAE.pdf

Motivation:

Test #1 is used to verify that the sensor is functioning correctly, by collecting and interpreting sensor data, which is critical. It also aids in understanding how to most effectively use the sensor and set it up (ie. signal conditioning, orientation etc.).

Procedure:

Test #1: The sensor's sensitivity is 59 Guass at a minimum of 18 mm as stated in the datasheet. Using a Neodynium disc shaped magnet with diameter 7.5mm and 2.5mm thick, which has a field strength of 142 Guass at 10mm (http://www.indigo.com/magnets/gphmgnts/Nd-disc-magnet-strength-specifications.html) we can verify that the sensor can detect the magnet and find out the max range of detection for these particular magnets. We measure the sensor's output using an oscilloscope.

Distance (mm)	Expected (V)	Reading (V)	Test Result
10	sinking output (0)	0	Pass
15	sinking output (0)	0	Pass
18	sinking output (0)	0	Pass
20	sinking output (0)	0	Pass
22	sinking output (0)	0	Pass
25	!(sinking output (0))	211	Pass
30	!(sinking output (0))	218	Pass

Discussion: As we can see from the above test we must have the magnets positioned at a range less than 20mm from the sensor for optimal detection.

4. Beagleboard Testing

4.1 Beagleboard Serial Module

Updates: The design has been modified eliminating the communication from the Arduino to the BeagleBoard. This is due to the fact that the whole obstacle detection system has been shifted from specialized sensor hardware to the camera, which is interfaced with the Beagleboard. Removing this module has eliminated the previous errors we encountered when the Arduino was communicating with the BeagleBoard. With these changes a driver module that simulates communication from the Arduino to the Beagleboard no longer needs to be used in the testing of the Beagleboard serial module.

Motivation:

The Beagleboard serial communication module needs to be tested separately from the Arduino serial module, it is necessary to isolate the modules when testing to determine the origin of errors in the system. In this test we rely only on the Beagleboard side module, we also create a driver module that is based on the Beagleboard module to simulate the Arduino, any errors that we encounter will have been created by the Beagleboard serial communication. Without the correctness of this module we fail to meet many of our requirements, the entire system depends on reliable communication between the Beagleboard and Arduino. The results of the path choice module must be forwarded to the Arduino via the serial module verbatim. If it can be shown that there are no problems with this module then our attention can be focused on correcting the Arduino side serial module. In this test we want to verify two aspects of this module, we want to make sure that the module is transmitting what it should transmit and we want to make sure that the module receives the correct values that are actually sent to it.

Procedure:

In this test we have the Beagleboard connected to a computer using a standard serial cable, as we want to reduce any problems that could be encounter with software included with the serial to USB cable. Both the Beagleboard and the computer will be running Linux. On the Beagleboard the actual serial communication module is used, and on the computer a modified version of the Beagleboard serial module is used. This driver module was modified such that the transmit packet is consistent with the receive packet in the Arduino side serial module. Then a program is made such that it randomly generates packets with values that are consistent with the requirements. These packets are then transmitted with the serial communication module. This module's verification consists of the following test.

Test #1: It must be shown that the serial module transmits what it should be actually be transmitting. The test is used to verify the send functionality of the module. We save all of the initial values that should be sent to a file, and then we transmit the value. With the use of a serial sniffer on the Beagleboard we see the actual value that was sent over serial, this is saved to a file. The difference of the two files is compared to see any inconsistencies. In this test we transmit 10000 packets and analyze the results.

Beagleboard → Driver Module

Desired Value to be Sent	Actual Value Sent	Actual Value Received	Test result
104	104	104	Pass
31	31	31	Pass
84	84	84	Pass
29	29	29	Pass
3	3	3	Pass

Discussion:

The complete results were not included for space but we can see that over 10000 packets, in all values combined we had about a 2.3% inconsistency. The change in design has addressed the previous issues encountered with the serial communication modules

Test #2: Communication from the Arduino to the Beagleboard has been eliminated, as a result the second test has been removed. With this change in design we no longer experience the serial communication errors encountered in this module.

Potential Problems:

Using a driver module to test the serial communication may have entered some new errors in the system, or it could also hide the effect of other errors in the actual system. Using the same serial module on both sides may amplify the effect of some errors as they could possibly be counted twice. If we send a packet that contains an error then it will also be received and counted as an error, so we must be careful when checking the results.

4.2 Path Choice Module

Testing was initially held in the modules path_choice and path_correction. These modules were designed to fulfill the requirement that the vehicle be able to avoid obstacles and for the vehicle to be able to stay within the same lane or change lanes at will (Requirements: SRS 3.3.7, SRS 3.3.14, SRS 3.3.8, SRS 3.3.3, SRS 3.3.16, SRS 3.3.6). The testing for these modules consisted of a set of predetermined cases, where, for example in the path choice module consisted in a series of obstacle configurations. These were broken down into an [X , X , X] arrangement, where X is the presence or absence of an obstacle in any particular lane, covering all 8 possible scenarios.

With each of these scenarios the tests were broken up into 3 blocks of behavior, depending on which lane the vehicle is in at that particular time. Depending on where the vehicle was, the distance to the obstacle was taken or not taken into account for the decision. This allowed for fast choice of a new lane, and more complex maneuvers with multiple obstacles.

For the test cases of the path_correction modules, this module provided the ability to make minor corrections within a lane to create a smooth lane following behavior. It also had the additional functionality to override the lane following and trigger a lane change if path_choice so requested it. Testing for this module consisted of verification of correction values to be of reasonable magnitude, and also in proportion the deviation from the center of the lane. Furthermore, inputs from path_choice were also simulated in order to verify that the lane change angle was correct magnitude and direction.

Lastly, in implementation it was found that these modules were best merged into a larger module in the form of a class. This module was implanted as pathfollow. However, testing has not been updated for these new modules, because of the particular dynamic nature of them. The previous test cases still hold validity.

Function Table

The following is the function table describing the behavior of the system given different obstacle configurations. We have broken them up into cases depending which lane we are in, with the subsequent breakdown being the different existence of obstacles variations, 8 in total. X's represent scenarios in which the distances between obstacles are not take into account for decision making (don't cares), the rightmost column represents the lane change (if any).

The naming convention we agreed upon was one relative to the cars position, where assuming the car is in the middle lane, lane 1 is the lane to the left and lane 3 is the one to the right. This convention stays the same regardless of clockwise or counter clockwise direction around the track.

We have determined that this function table is both complete and disjoint because it covers all scenarios of obstacle configurations, and includes the distances that are of importance to its behavior.

Legend: CurLane = current lane we are in. ObsChk = existence of obstacles in lanes. OD# = distance to obstacle in lane #. NewLane = lane change, if different from CurLane car will move to that lane. 40 = 40 cm. DL = default lane.

CurLane	**ObsChk**	**OD1-OD2**	**OD2-OD1**	**OD2-OD3**	**OD3-OD2**	**NewLane**
1						
	0 0 0	X	X	X	X	*DL*
	0 0 1	X	X	X	X	*1*
	0 1 0	X	X	X	X	*1*
	0 1 1	X	X	X	X	*1*
	1 0 0	X	X	X	X	*2*
	1 0 1	X	X	X	X	*2*
	1 1 0	>40	X	X	X	*1*
		<40	X	X	X	*2*
	1 1 1	>40				*1*
		<40	<40	<40	<40	*Stop*

CurLane	**ObsChk**					
2						
	0 0 0	X	X	X	X	*DL*
	0 0 1	X	X	X	X	*2*
	0 1 0	X	X	X	X	*DL*
	0 1 1	X	X	X	X	*1*
	1 0 0	X	X	X	X	*2*
	1 0 1	X	X	X	X	*2*
	1 1 0	X	X	X	X	*3*
	1 1 1	X	>40	X	X	*2*
		X	X	>40	X	*2*
		>40	X	X	X	*1*
		X	X	X	<40	*3*
		<40	<40	<40	<40	*stop*

CurLane	**ObsChk**					
3						
	0 0 0	X	X	X	X	*DL*
	0 0 1	X	X	X	X	*2*
	0 1 0	X	X	X	X	*3*
	0 1 1	X	X	X	>40	*3*
		X	X	X	<40	*2*
	1 0 0	X	X	X	X	*3*
	1 0 1	X	X	X	X	*2*
	1 1 0	X	X	X	X	*3*
	1 1 1	X	X	X	>40	*3*
		<40	<40	<40	<40	*stop*

4.3 Path Correction Module

(Requirement: SRS 3.3.7) This module was designed as a proportional controller in which the deviations "Distance to center" is used as a multiplier for the correcting factor sent to the wheels of the car. In first instance this system behaved with great stability at low speeds, providing very smooth correction as it went around the track:

CorrectionAngle = k*DistancetoCenter(0)

However at high speeds the refresh rate became too slow for the upcoming points to be processed and calculated for a "Distance to center". We therefor decided it would suit our needs to look head another 2 points and do a weighted average to improve higher speed performance, hence it became.

CorrectionAngle = k*[DistancetoCenter(0)*0.7 + DistancetoCenter(1)*0.2 + DistancetoCenter(2)*0.1]

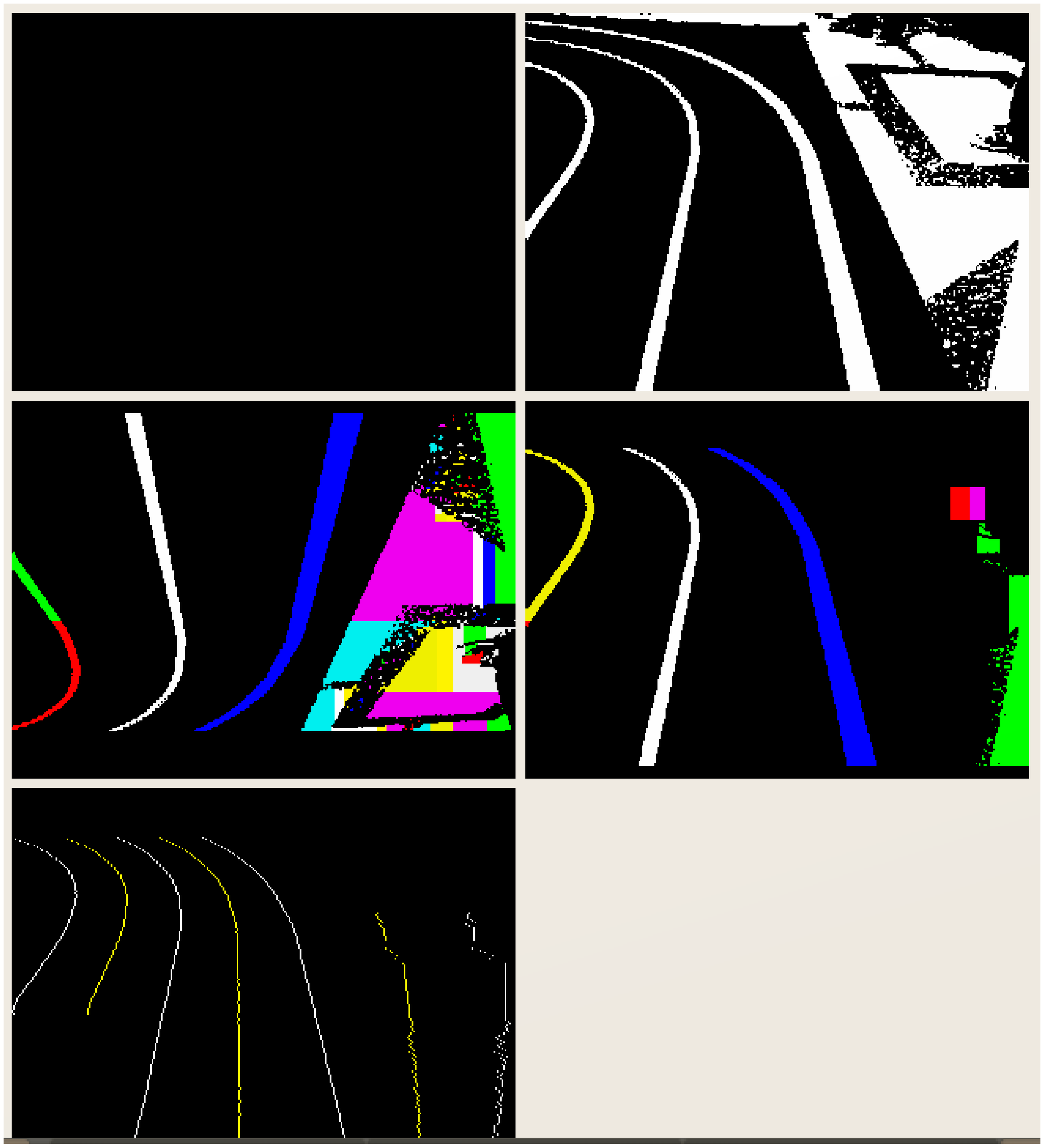
This configuration provides smooth correction at higher speeds up to a tested lap time of 20 seconds on the outer lane.

4.4 Camera Module – Lane Detection

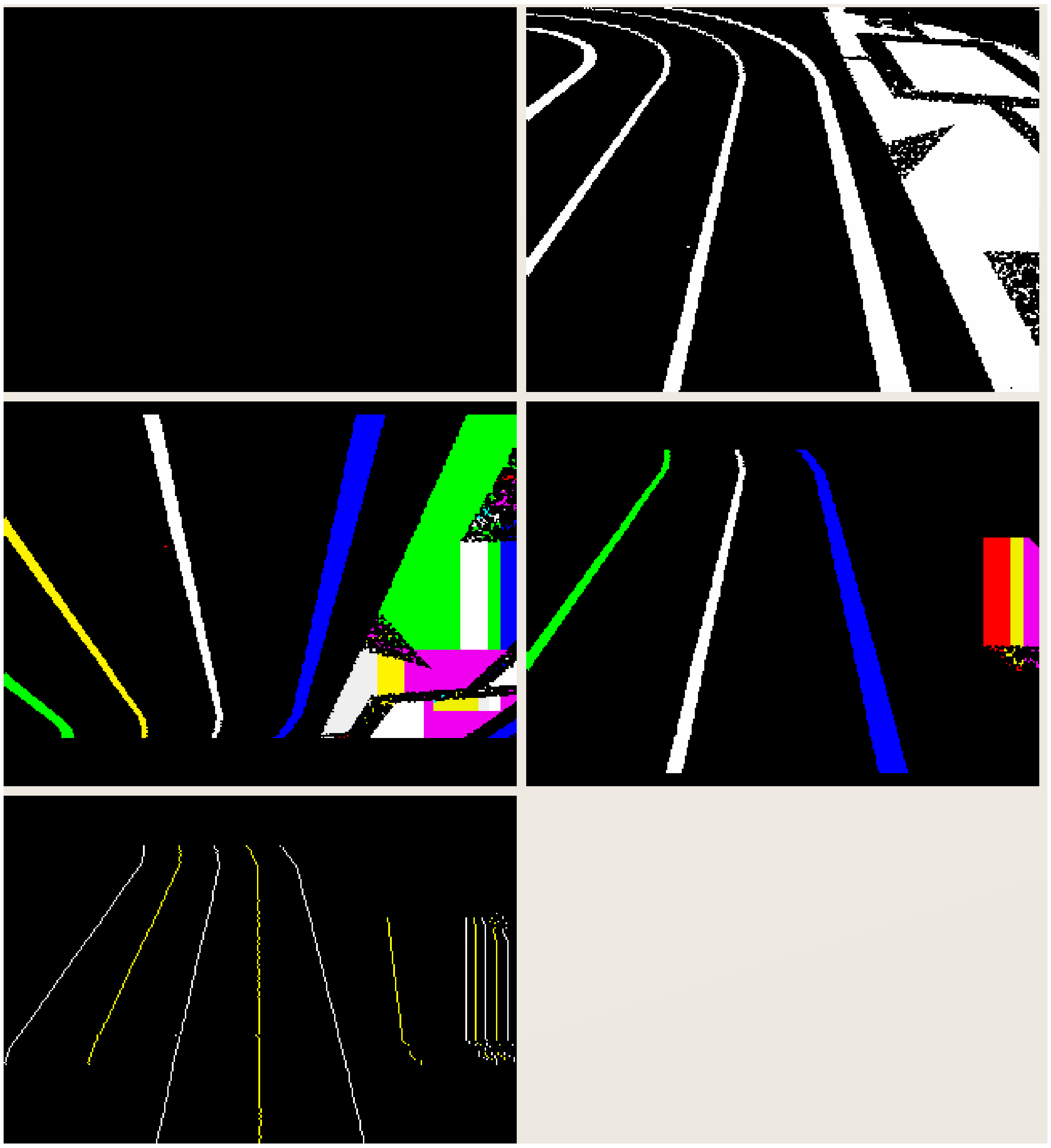
Motivation: This is a key component of the system. One of the main requirements of the system is to be able to detect all lanes of the track. To verify we are fulfilling this requirement this test is necessary. If successful we can be sure that the system will be able to detect lanes in all scenarios. (Requirement: SRS 3.3.4)

Procedure:
A special testing program has been written for this portion of the V&V. It allows a laptop to run the same code as on the beagleboard with the added benefit of displaying all stages of the image processing and lane detection. The vehicle will be placed on the track in all 6 scenarios for lane following (Lane 1, 2, 3 – clockwise and counter clockwise) and any noise or undesired events will be noted below.

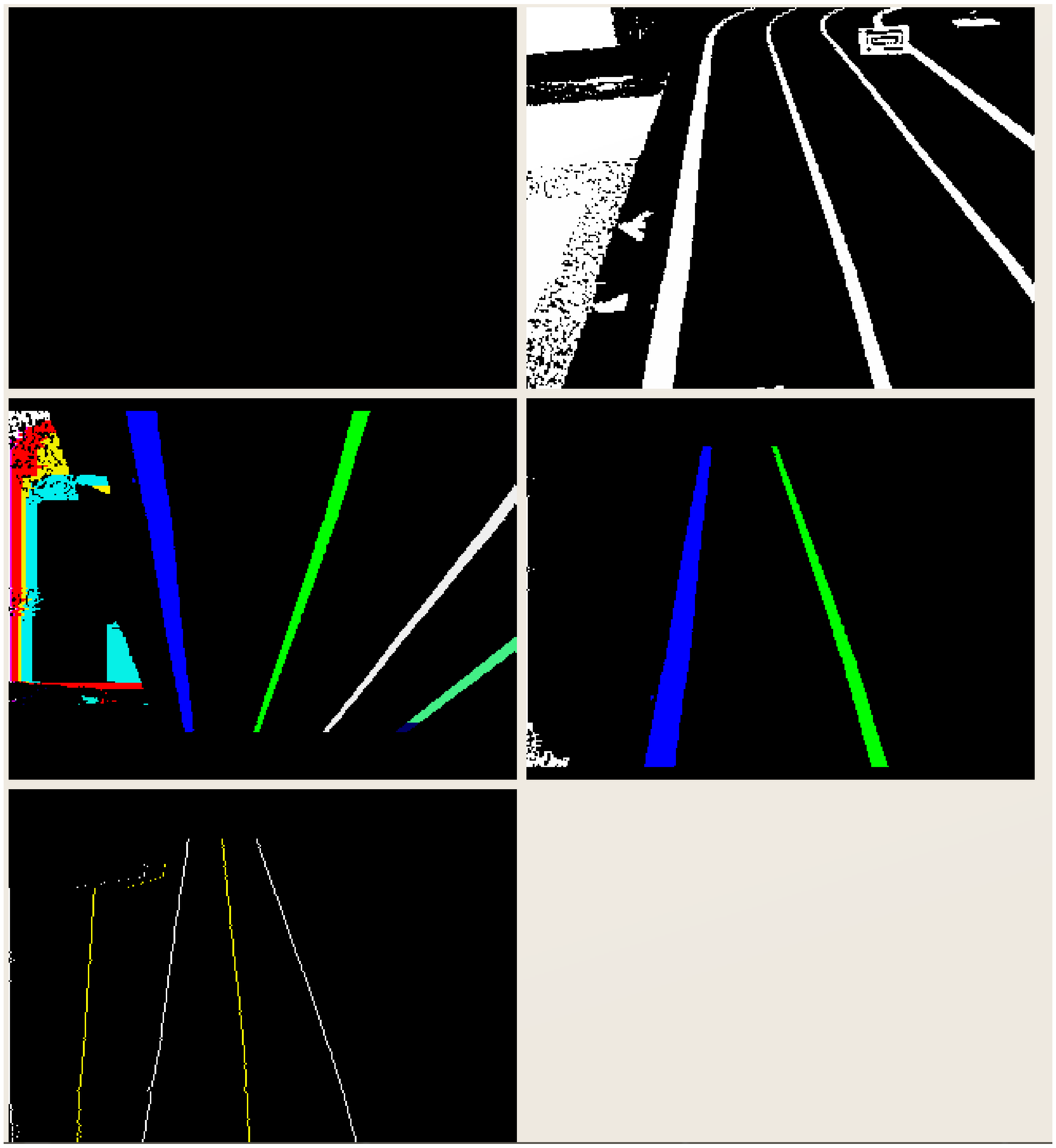
The below image shows the reflectivity of the floor adding false positives to the lane detection. The filtering system removes most of the noise from the image prior to lane detection though. Note that the white lines in the final image are lanes and the yellow lines are the midpoints between two lanes.



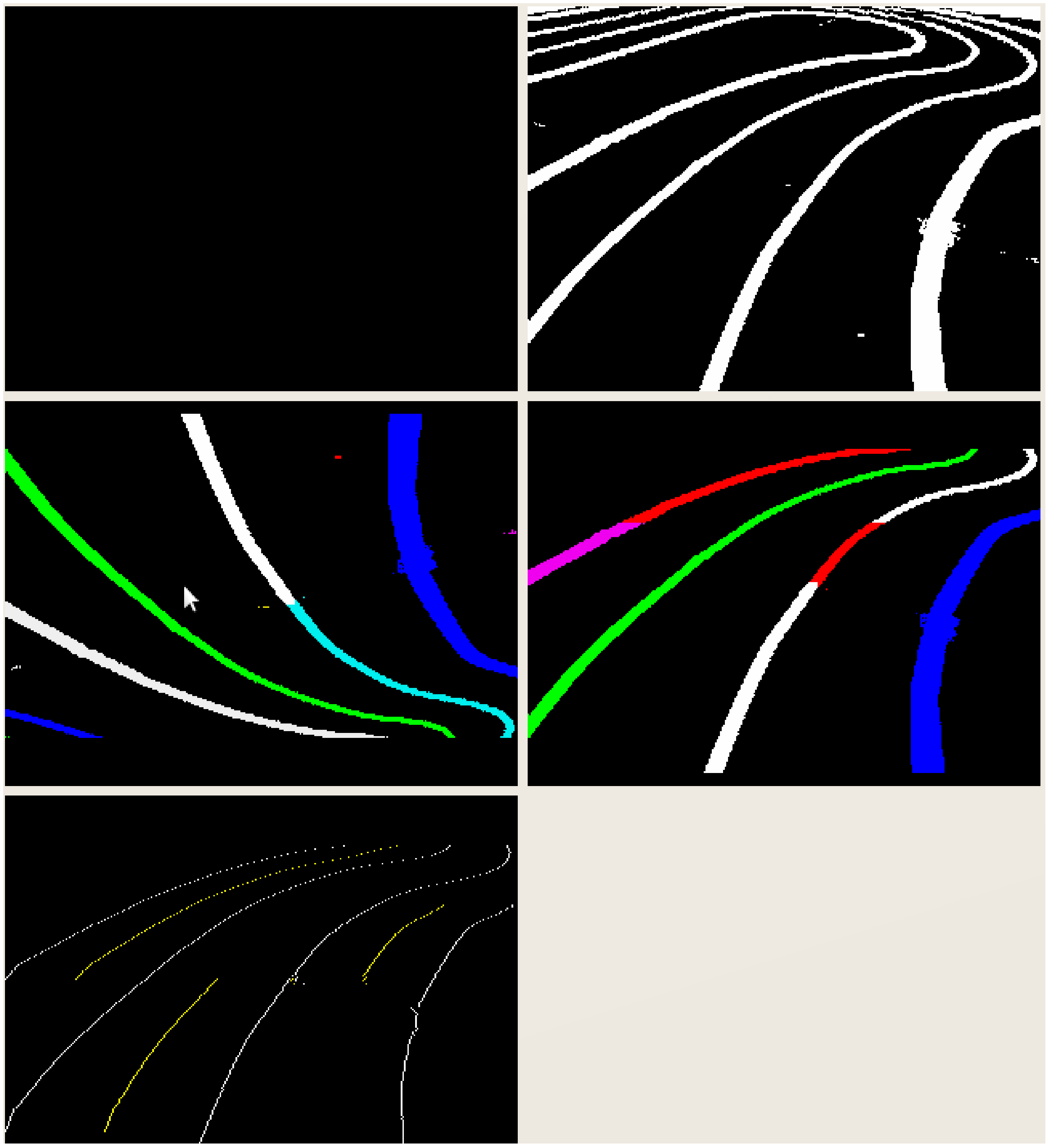
The third set of midpoints from the left looks very similar to a proper lane.



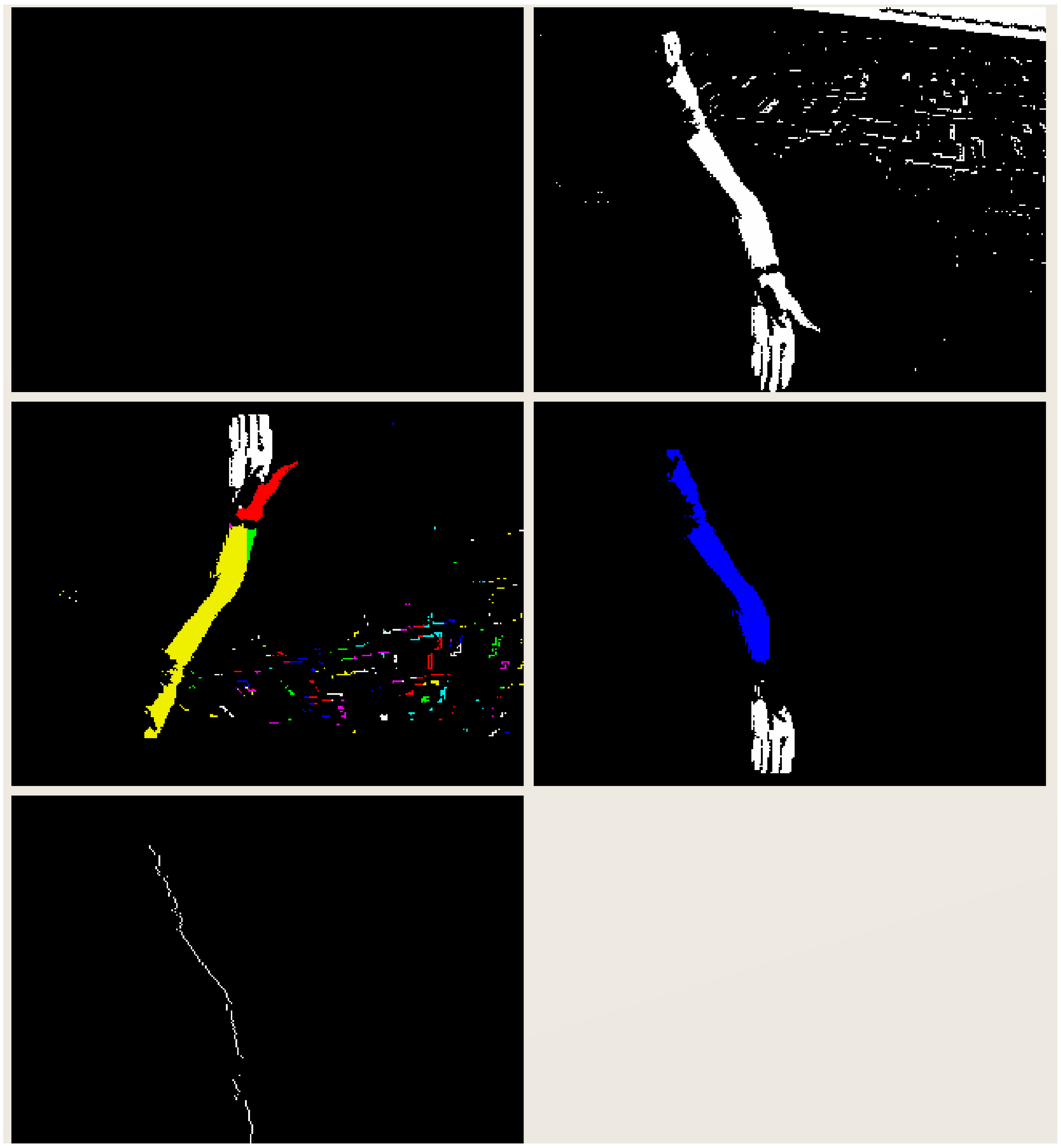
The below image shows another example of noise looking very much like a valid lane.



The below image shows how the tape joining the two sections of carpet shows up as noise.



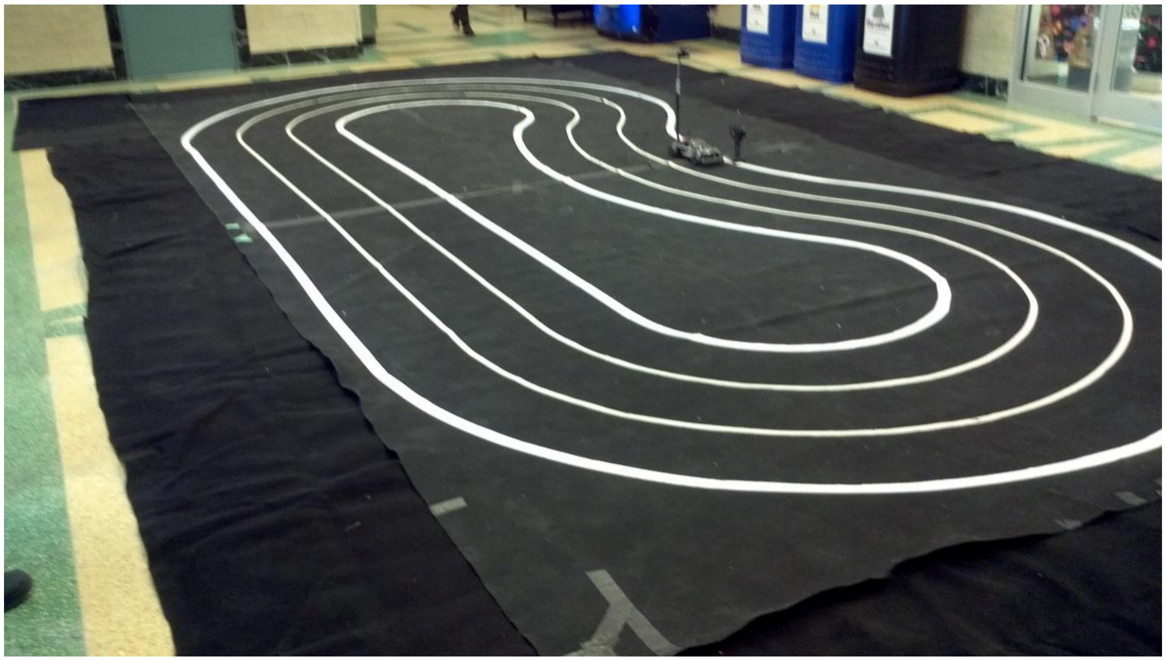
An example of how Steven Back's arm is detected by the camera. This causes potential problems as the camera automatically adjusts its picture based upon what it sees. By having other objects in the frame, different adjustments are made changing the picture. Since the camera tries to show Steven Back's arm, it adjusts various parameters (gain, white balance) to see it better, revealing potentially other undesirable features. His arm is also seen as a lane and is processed accordingly.



Potential Problems:

The main potential problems are that the image filtering allows certain blobs of noise through which causes them to be processed as potential lanes. Likewise, the auto adjustment of settings by the camera causes lighting conditions and outside environment conditions to cause inconsistencies in the image processing.

Test 2: In this test the carpet was surrounded by a 2 foot extension of black carpet. The image processing combined with a more refined filter eliminated all blobs that were considered noise.

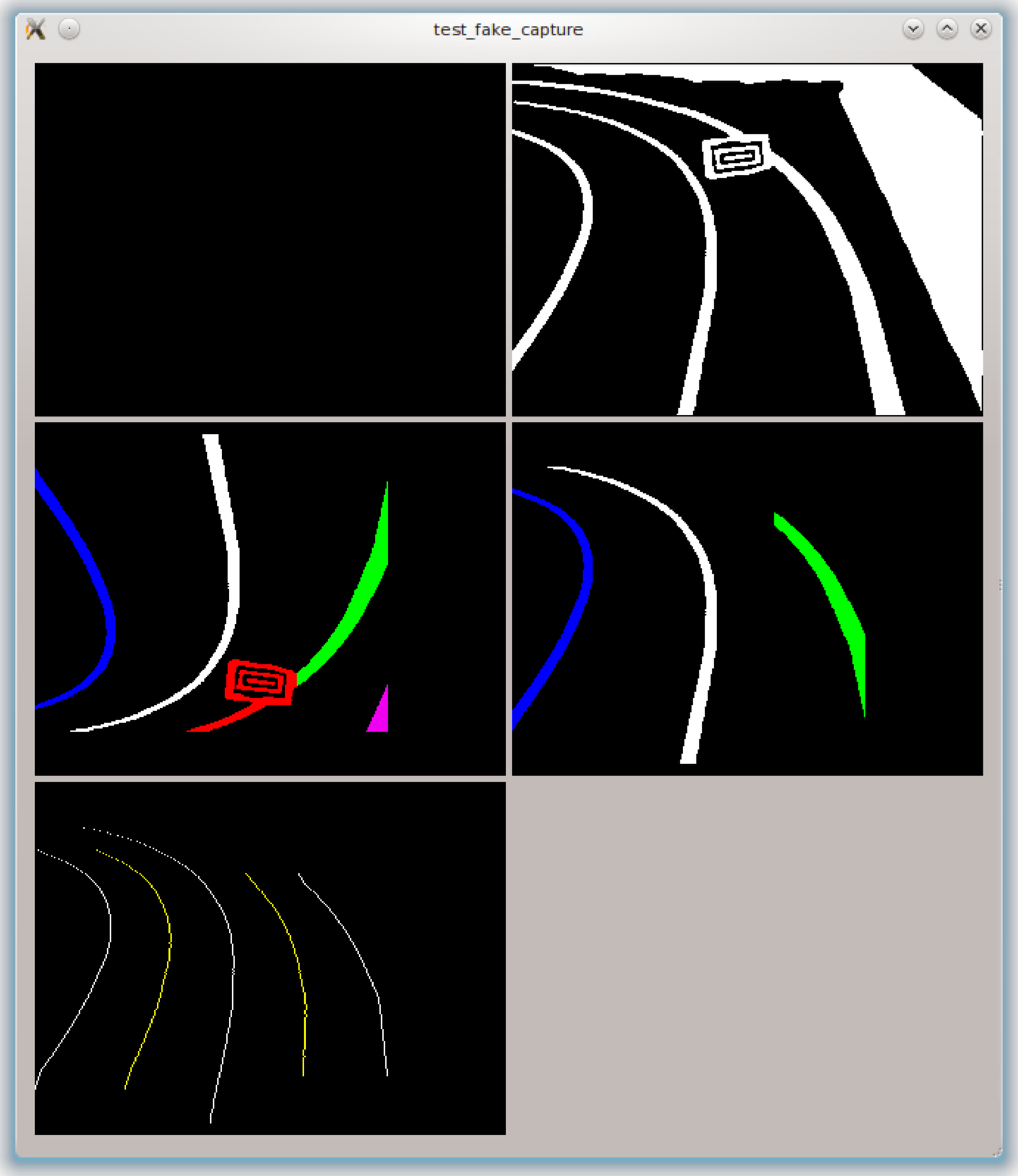


4.5 Camera Module – Obstacle Detection

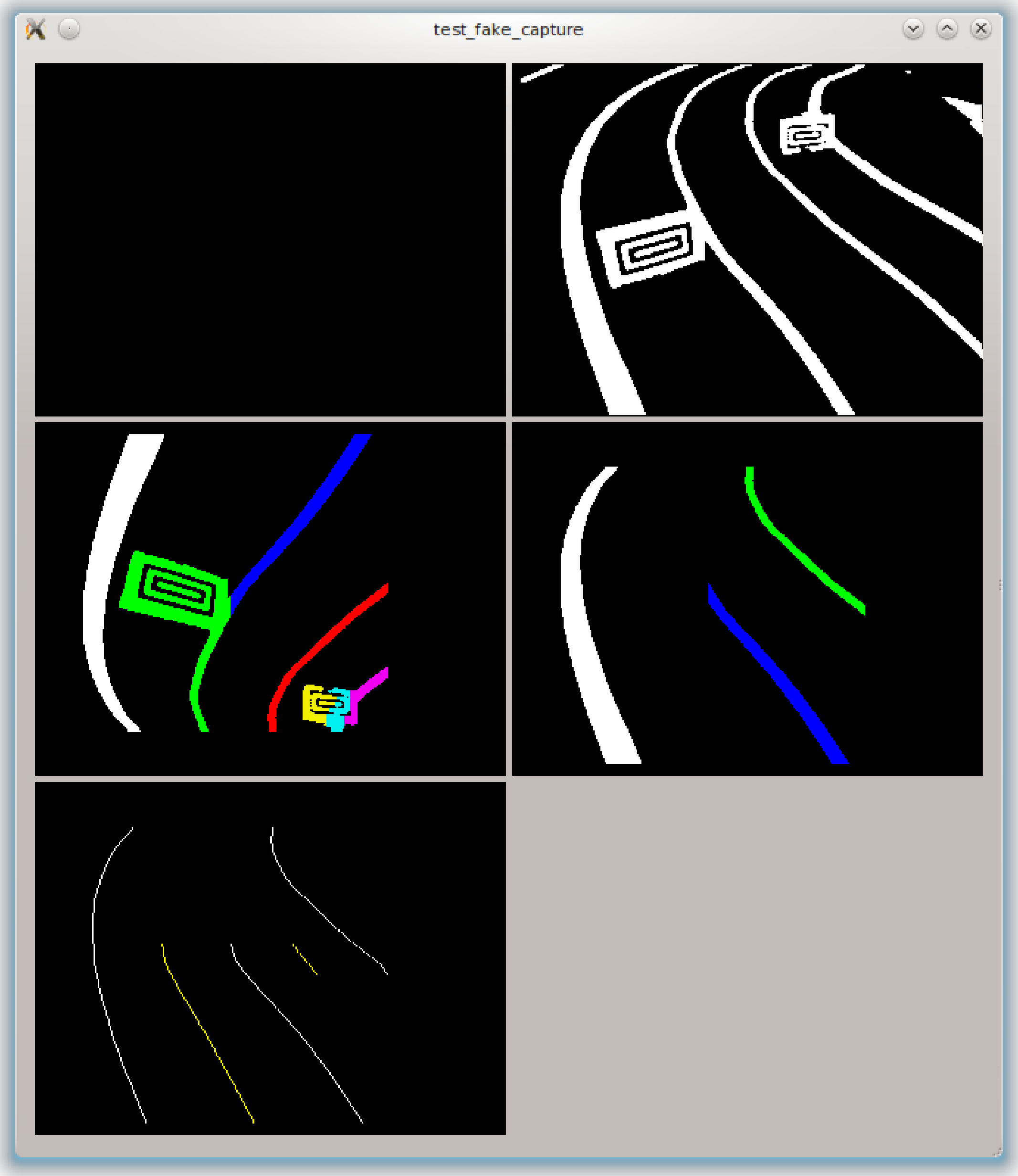
Motivation: This is a key component of the system. One of the main requirements of the system is to be able to detect all obstacles on the track. To verify we are fulfilling this requirement this test is necessary. If successful we can be sure that the system will be able to detect lanes in all scenarios. (Requirement: SRS 3.3.13)

Procedure:
A special testing program has been written for this portion of the V&V. It allows a laptop to run the same code as on the Beagleboard with the added benefit of displaying all stages of the image processing and obstacle detection. The vehicle will be placed on the track in all obstacle scenarios for obstacle detection and any undesired events will be noted below.

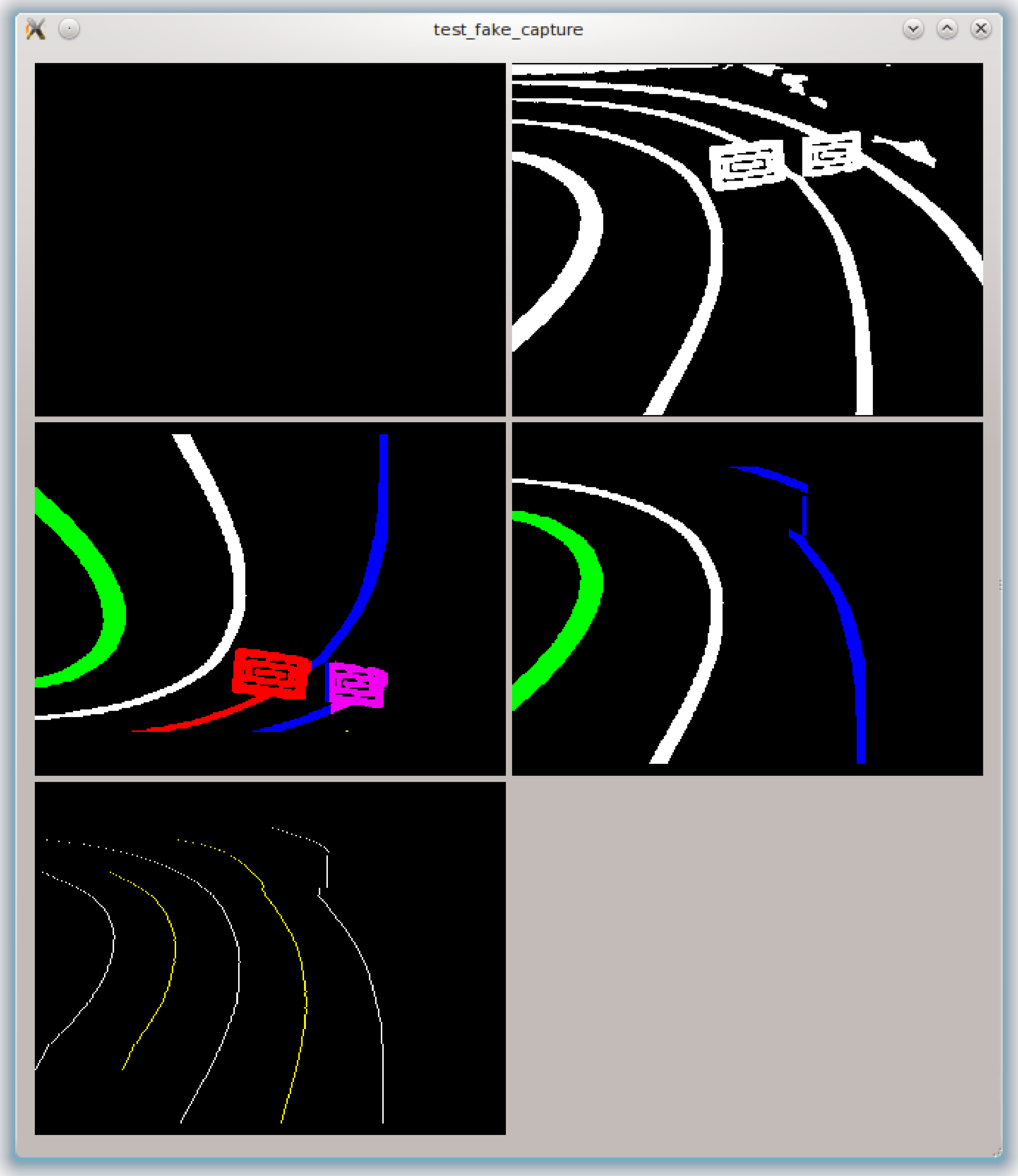
This image shows an obstacle being properly detected on the track. It is one solid colour throughout. The only current issue is that the obstacle continues along the boundary of the track. Instead it should finish and the track continues.



While the first obstacle is fine, the second obstacle, having a broken section, is identified as several blobs. The track is also cut off, again. This could be fixed by simply assuming any obstacles close together are the same, and that is implemented elsewhere.



When two obstacles are close, they again interfere incorrectly. This requires more filtering when deciding to what blob a pixel belongs.



5. Simulations

The purpose of the simulations was to get a good idea about the logic behind machine vision and test different methods to accomplish the goals of the project: lane detection and obstacle avoidance.

The simulations were primarily done on MATLAB, but a specialized computer vision program called Roborealm was used as well. MATLAB provided the ability to perform simulations on the lane following algorithms while avoiding any details in the actual implementation of the algorithms. There were two main aspects of the simulations, the first was to verify the motion of the car by simulation the servo control and the second was to simulate lane detection on actual images of the track to determine if the midpoints were being properly calculated.

5.1 Turning Servo Control (MATLAB)

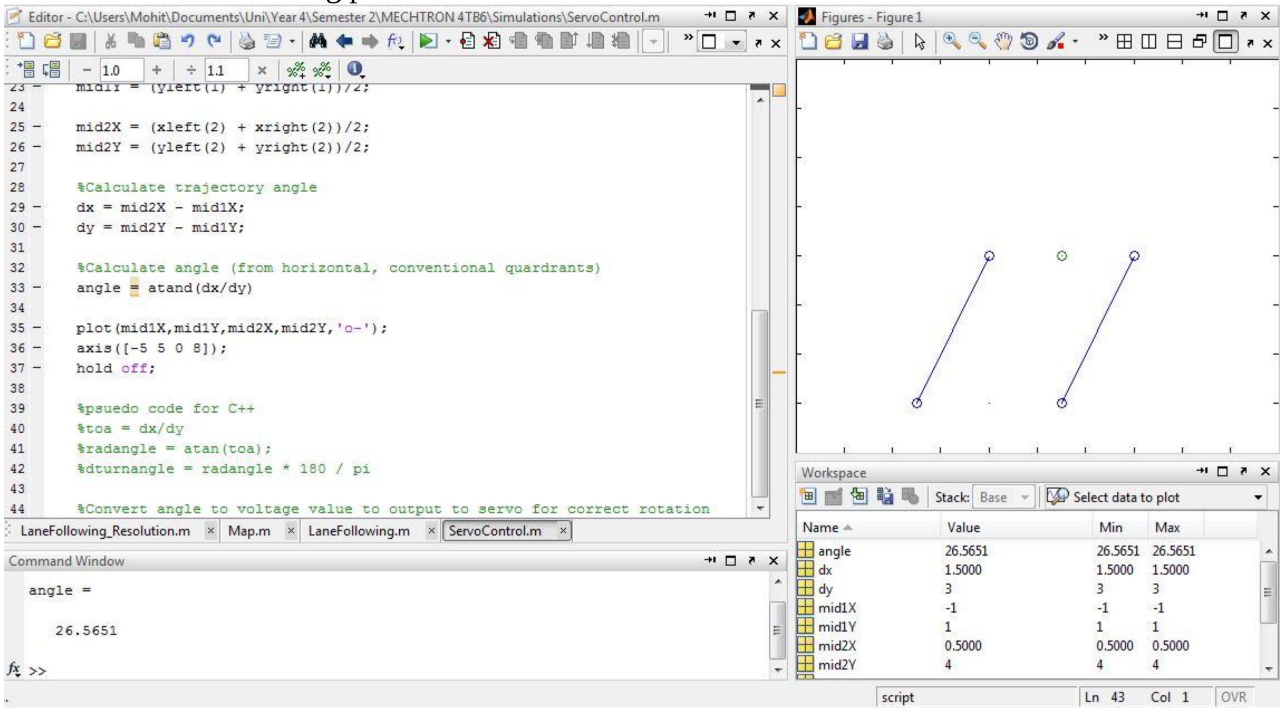
The first simulation done was a basic plot to calculate the angle the servo must turn to complete a segment of a curve.

The code:

```
clc;
clear;

%Simulation for Turning

%Left boundary
%Note: each interval = 10 cm
xleft = [-2.5 -1];
yleft = [1 4];

plot(xleft,yleft,'o-');
hold on;

%Right boundary
xright = xleft + [3 3]; %Lanes are 30 cm wide
yright = yleft;%[1 6];

plot(xright,yright,'o-');
hold on;

%Calculate midpoints
mid1X = (xleft(1) + xright(1))/2;
mid1Y = (yleft(1) + yright(1))/2;
```

```
mid2X = (xleft(2) + xright(2))/2;
mid2Y = (yleft(2) + yright(2))/2;

%Calculate trajectory angle
dx = mid2X - mid1X;
dy = mid2Y - mid1Y;

%Calculate angle (from horizontal, conventional quardrants)
angle = atand(dx/dy)

plot(mid1X,mid1Y,mid2X,mid2Y,'o-');
axis([-5 5 0 8]);
hold off;

%psuedo code for C++
%toa = dx/dy
%radangle = atan(toa);
%dturnangle = radangle * 180 / pi

%Convert angle to voltage value to output to servo for correct rotation
```

The result is the following plot and calculation

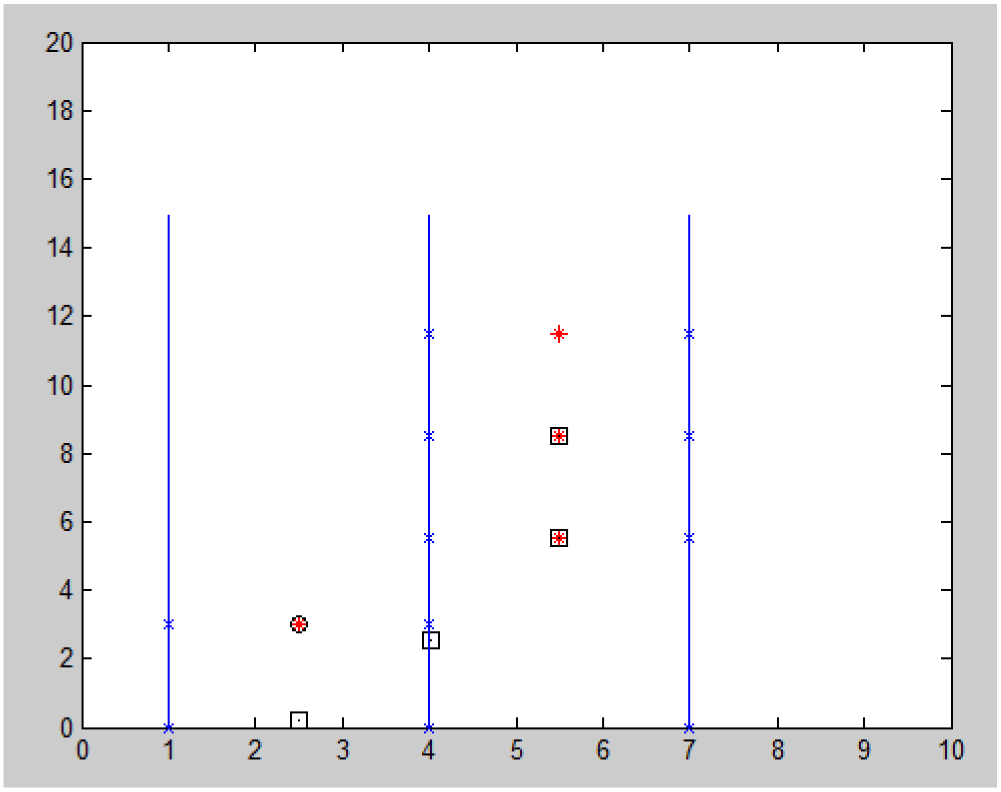


```
    mid1Y = (yleft(1) + yright(1))/2;

    mid2X = (xleft(2) + xright(2))/2;
    mid2Y = (yleft(2) + yright(2))/2;

    %Calculate trajectory angle
    dx = mid2X - mid1X;
    dy = mid2Y - mid1Y;

    %Calculate angle (from horizontal, conventional quardrants)
    angle = atand(dx/dy)

    plot(mid1X,mid1Y,mid2X,mid2Y,'o-');
    axis([-5 5 0 8]);
    hold off;

    %psuedo code for C++
    %toa = dx/dy
    %radangle = atan(toa);
    %dturnangle = radangle * 180 / pi

    %Convert angle to voltage value to output to servo for correct rotation
```

```
angle =

   26.5651

>>
```

Name	Value	Min	Max
angle	26.5651	26.5651	26.5651
dx	1.5000	1.5000	1.5000
dy	3	3	3
mid1X	-1	-1	-1
mid1Y	1	1	1
mid2X	0.5000	0.5000	0.5000
mid2Y	4	4	4

The results show us that the servo control is directing the vehicle towards the midpoints so can stay within a lane. When this is performed with multiple segments we see vehicles path remains within the lane boundaries.



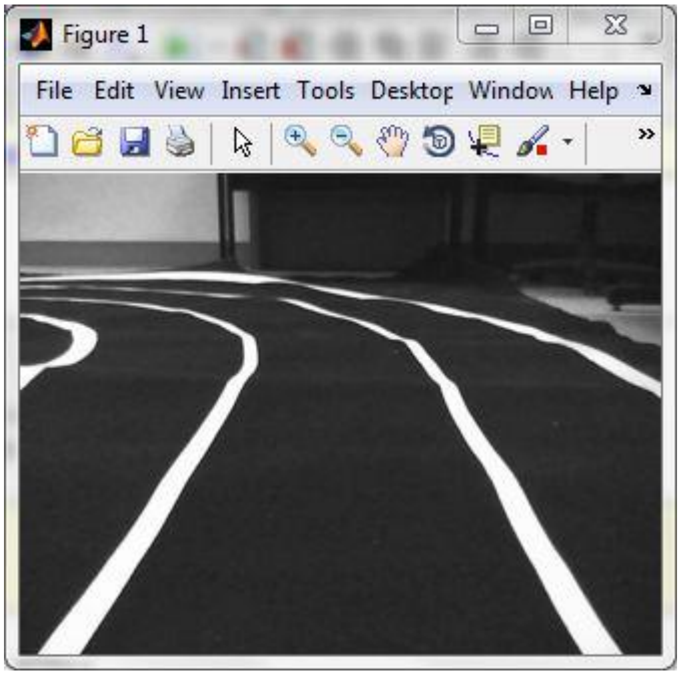
The circle is an obstacle and it was avoided.

5.2 Lane Detection (MATLAB)

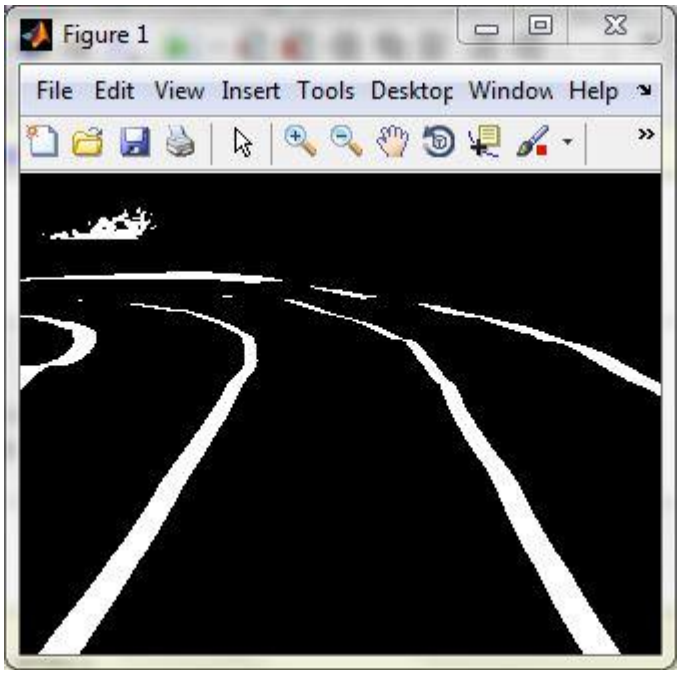
Using a spare webcam (not being used for the final design) the team took pictures of the track to analyze using MATLAB for lane detection.

The code along with resulting figures of each cell:

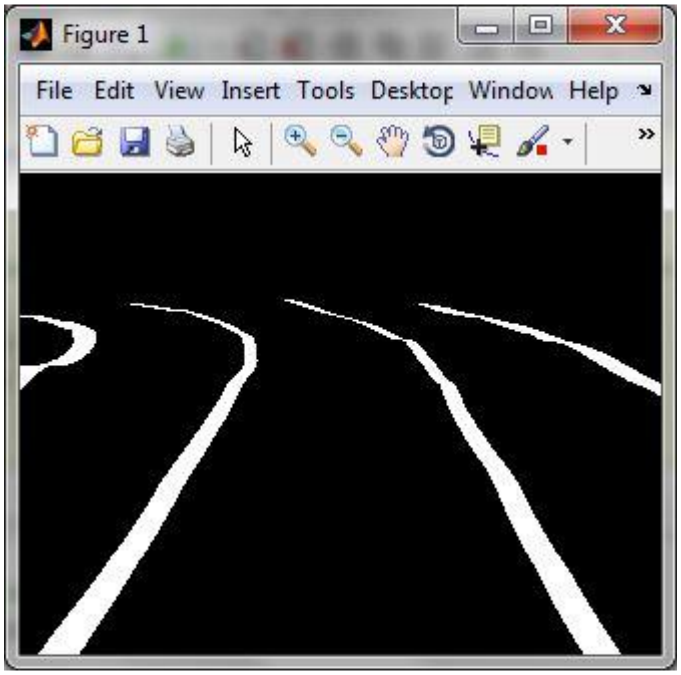
```
clear;
clc;
clf;
%%Lane Following
%%
%%Load Image
track = imread('track1.jpg');
track = imresize(track, [240 320]);
```



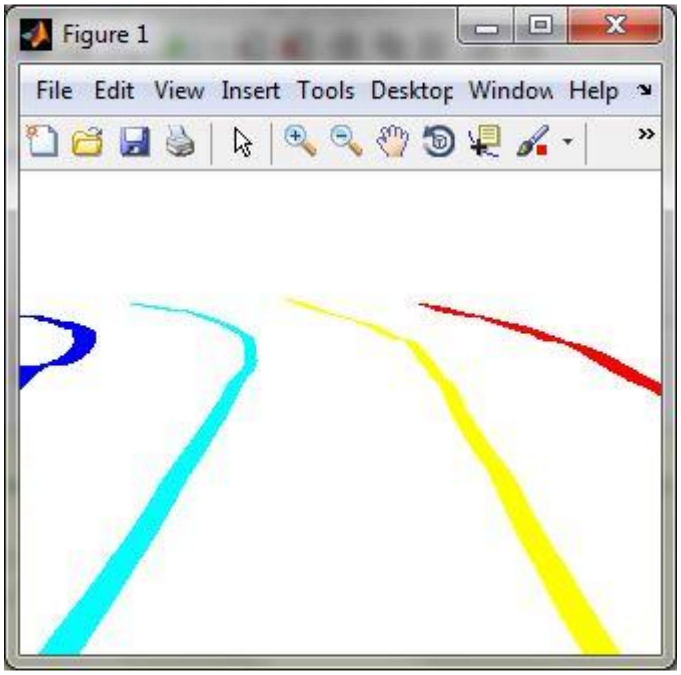
```
imshow(track);
%% Segment by Threshold
bwtrack = im2bw(track,.75);
```



```
imshow(bwtrack);

%% Clean image
lanes = bwareaopen(bwtrack, 370);
imshow(lanes);
```

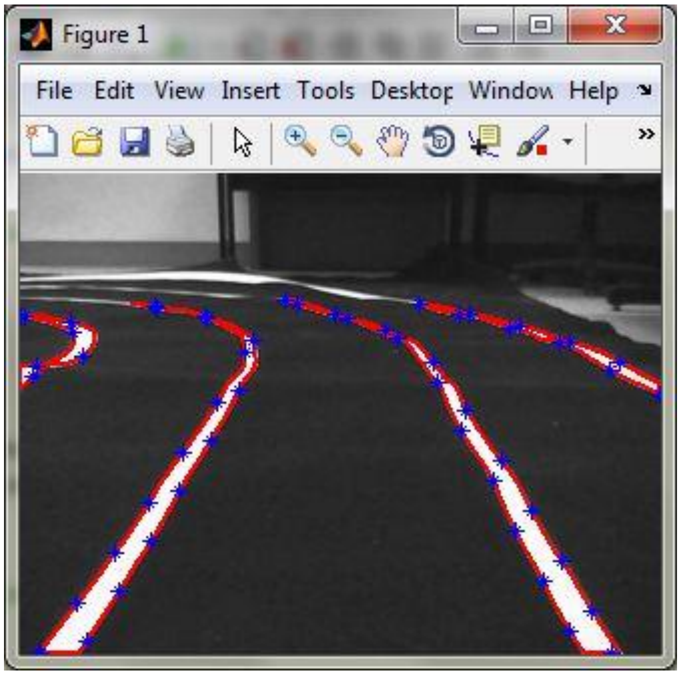


```
%% Find lanes
%L = store pixels of image as [0,1,2..] where 0 is no object and 1 is lane
%1 and so one
%B = array of pixel boundary locations
[B, L] = bwboundaries(lanes,'noholes');
numregions = max(L(:));
imshow(label2rgb(L));
```



```
%% Connected regions
stats = regionprops(L,'all');
%% Eccentricity
% Round object = 0, linear object = 1
shapes = [stats.Eccentricity];
keepers = find(shapes>.8);
%Note: instead of eccentricity in actual implementation we must use
%intensity difference to detect edges around the lanes.
%% Overlay lines over original image with edges
imshow(track);
hold on;

%Create outline
for index=1:length(keepers)
   outline = B{keepers(index)};
   line(outline(:,2),outline(:,1),'Color','r','LineWidth',2);
end
```

```
%Along mark points along edges
edX = [];
edY = [];
for i = 1:length(keepers)
   for edges = length(B{keepers(i),1})
      for bounds = 1:25:edges
         edX = B{keepers(i),1}(bounds,2);
         edY = B{keepers(i),1}(bounds,1);
         plot(edX, edY,'*','Color','b');
         hold on;
      end
   end
end
hold off;
```

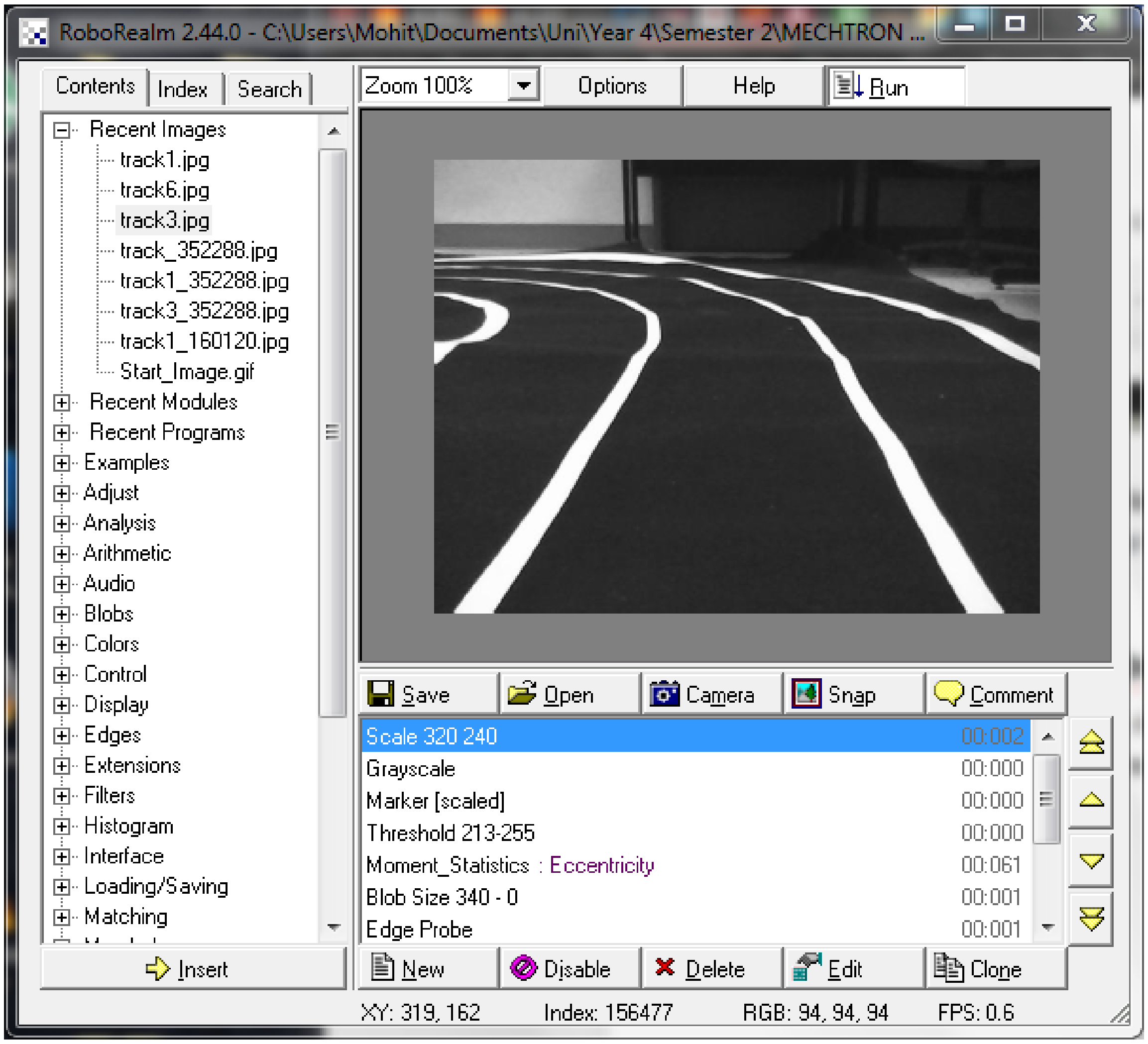


The result is the following image:

For later use, we can potentially connect the dots from one lane to adjacent lane and calculate the midpoint for the car to go to up ahead.

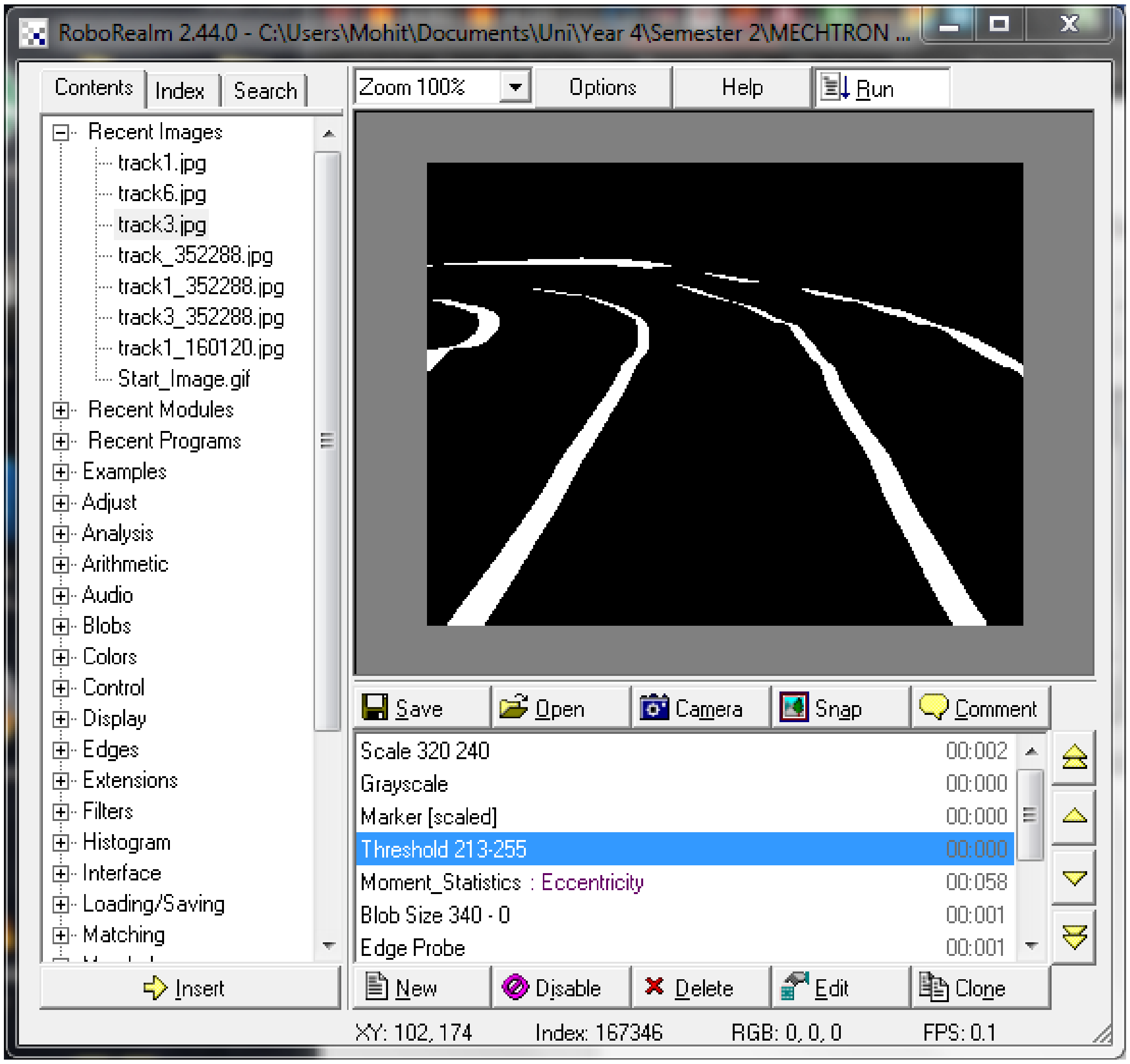
5.3 Lane Detection (RoboRealm)

RoboRealm provides a very intuitive GUI to apply some of the same features that required hard coding in MATLAB. Using the same image:



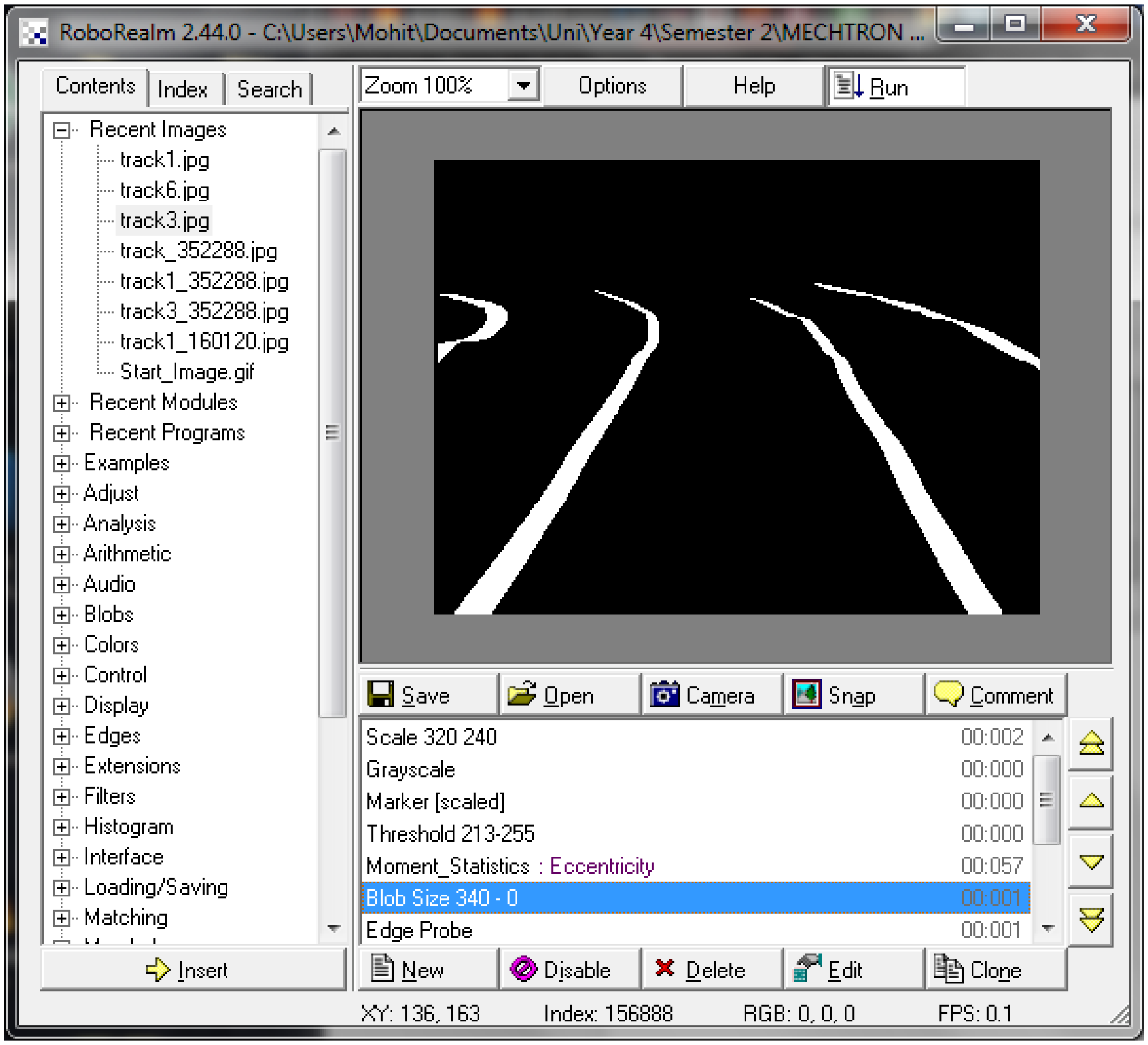
```
Scale 320 240                              00:002
Grayscale                                  00:000
Marker [scaled]                            00:000
Threshold 213-255                          00:000
Moment_Statistics : Eccentricity           00:061
Blob Size 340 - 0                          00:001
Edge Probe                                 00:001
```

```
XY: 319, 162    Index: 156477    RGB: 94, 94, 94    FPS: 0.6
```


RoboRealm 2.44.0 - C:\Users\Mohit\Documents\Uni\Year 4\Semester 2\MECHTRON ...

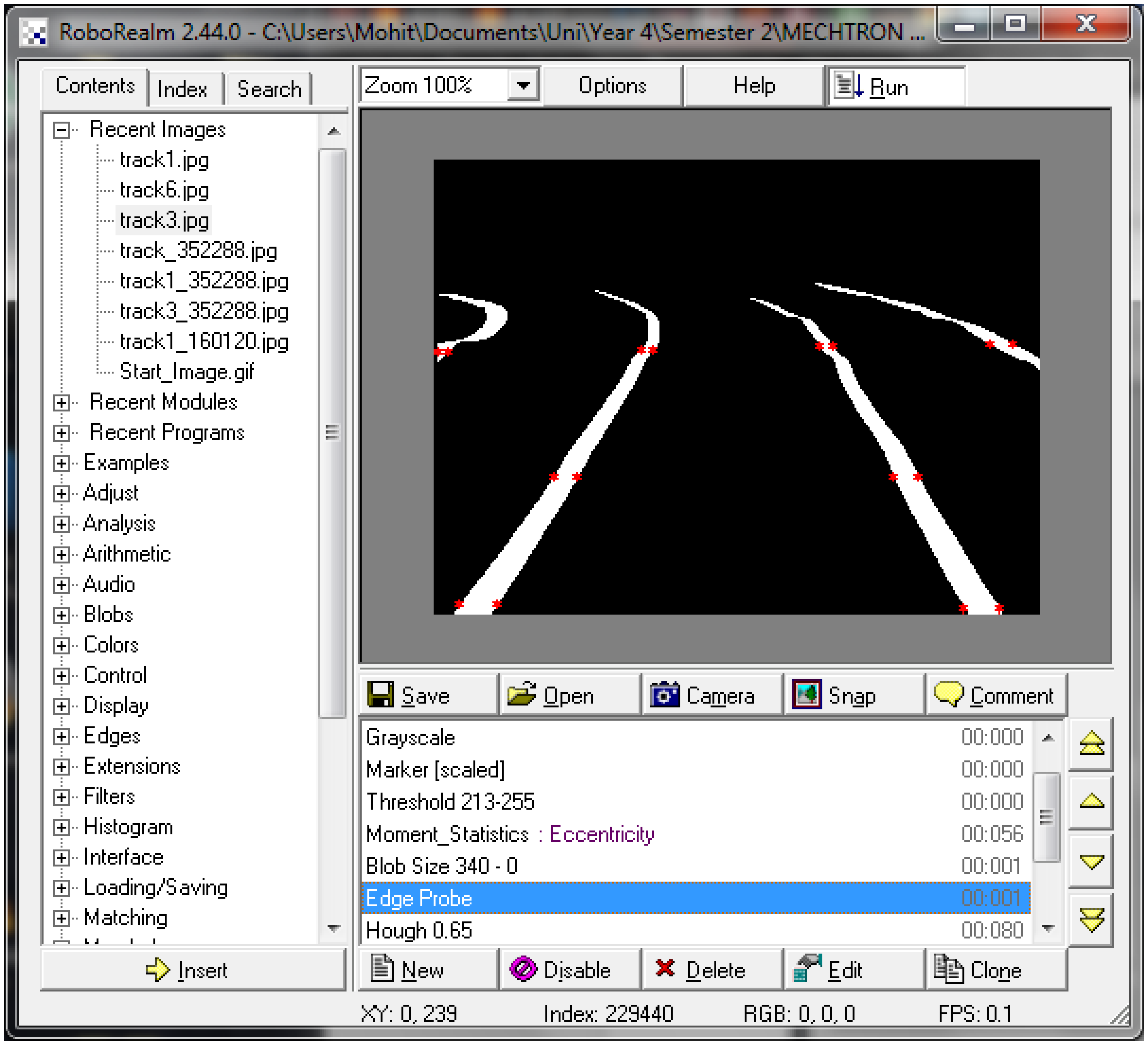
```
Scale 320 240                           00:002
Grayscale                               00:000
Marker [scaled]                         00:000
Threshold 213-255                       00:000
Moment_Statistics : Eccentricity        00:058
Blob Size 340 - 0                       00:001
Edge Probe                              00:001
```

```
XY: 102, 174     Index: 167346     RGB: 0, 0, 0     FPS: 0.1
```



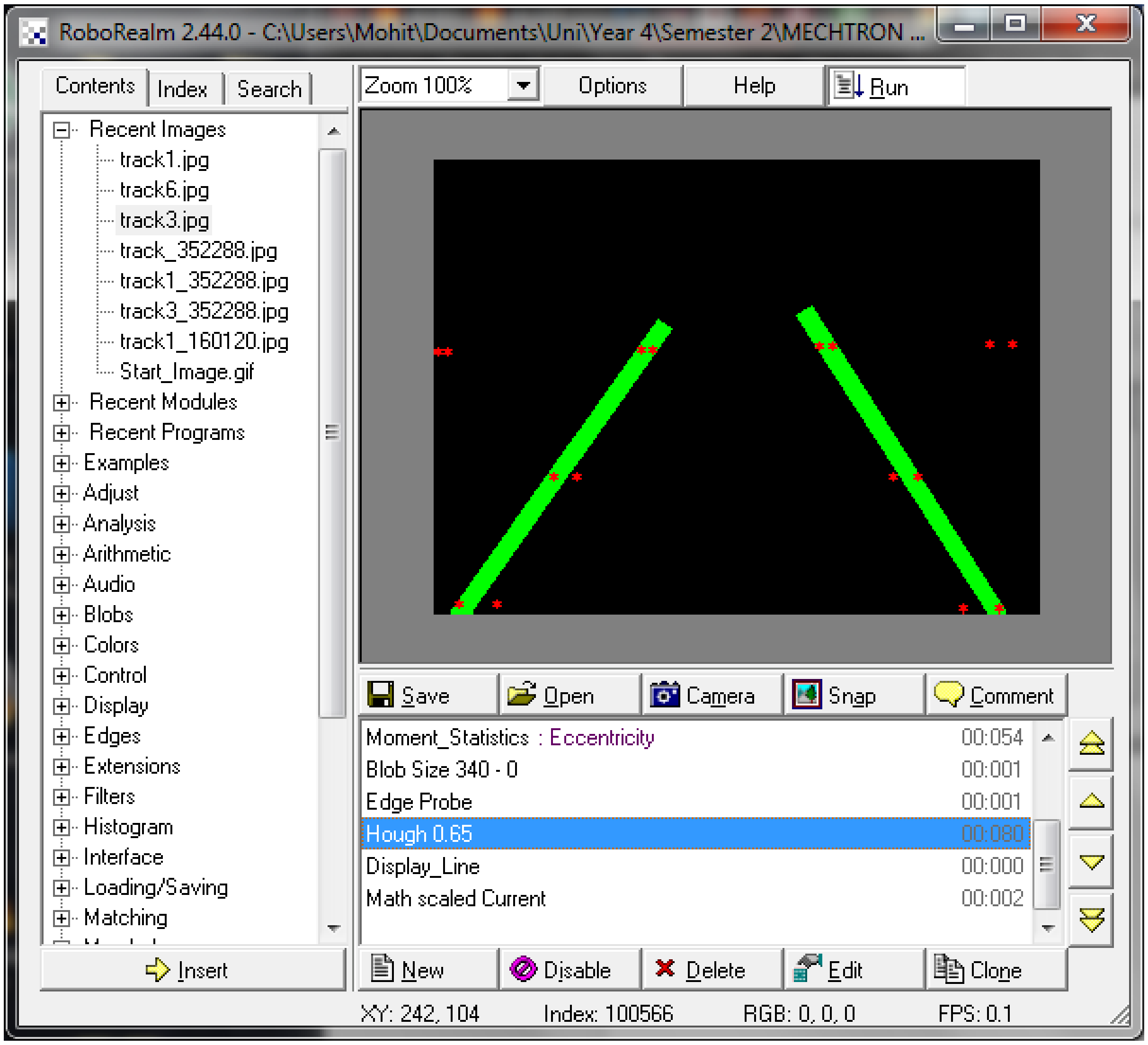
```
Scale 320 240                              00:002
Grayscale                                  00:000
Marker [scaled]                            00:000
Threshold 213-255                          00:000
Moment_Statistics : Eccentricity           00:057
Blob Size 340 - 0                          00:001
Edge Probe                                 00:001
```

```
XY: 136, 163    Index: 156888    RGB: 0, 0, 0    FPS: 0.1
```



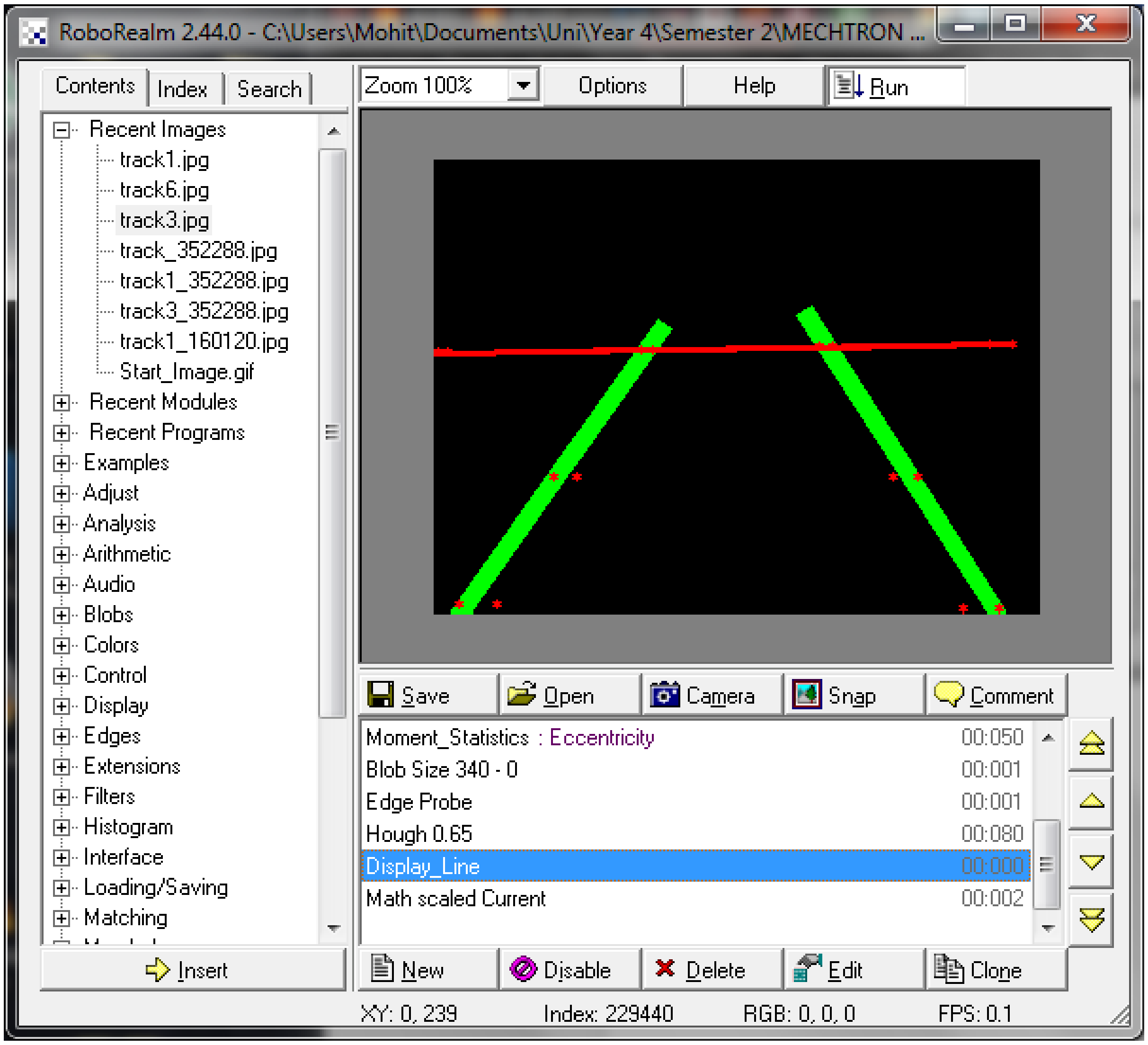
```
Grayscale                          00:000
Marker [scaled]                    00:000
Threshold 213-255                  00:000
Moment_Statistics : Eccentricity   00:056
Blob Size 340 - 0                  00:001
Edge Probe                         00:001
Hough 0.65                         00:080
```

XY: 0, 239 Index: 229440 RGB: 0, 0, 0 FPS: 0.1



```
Moment_Statistics : Eccentricity     00:054
Blob Size 340 - 0                    00:001
Edge Probe                           00:001
Hough 0.65                           00:080
Display_Line                         00:000
Math scaled Current                  00:002
```

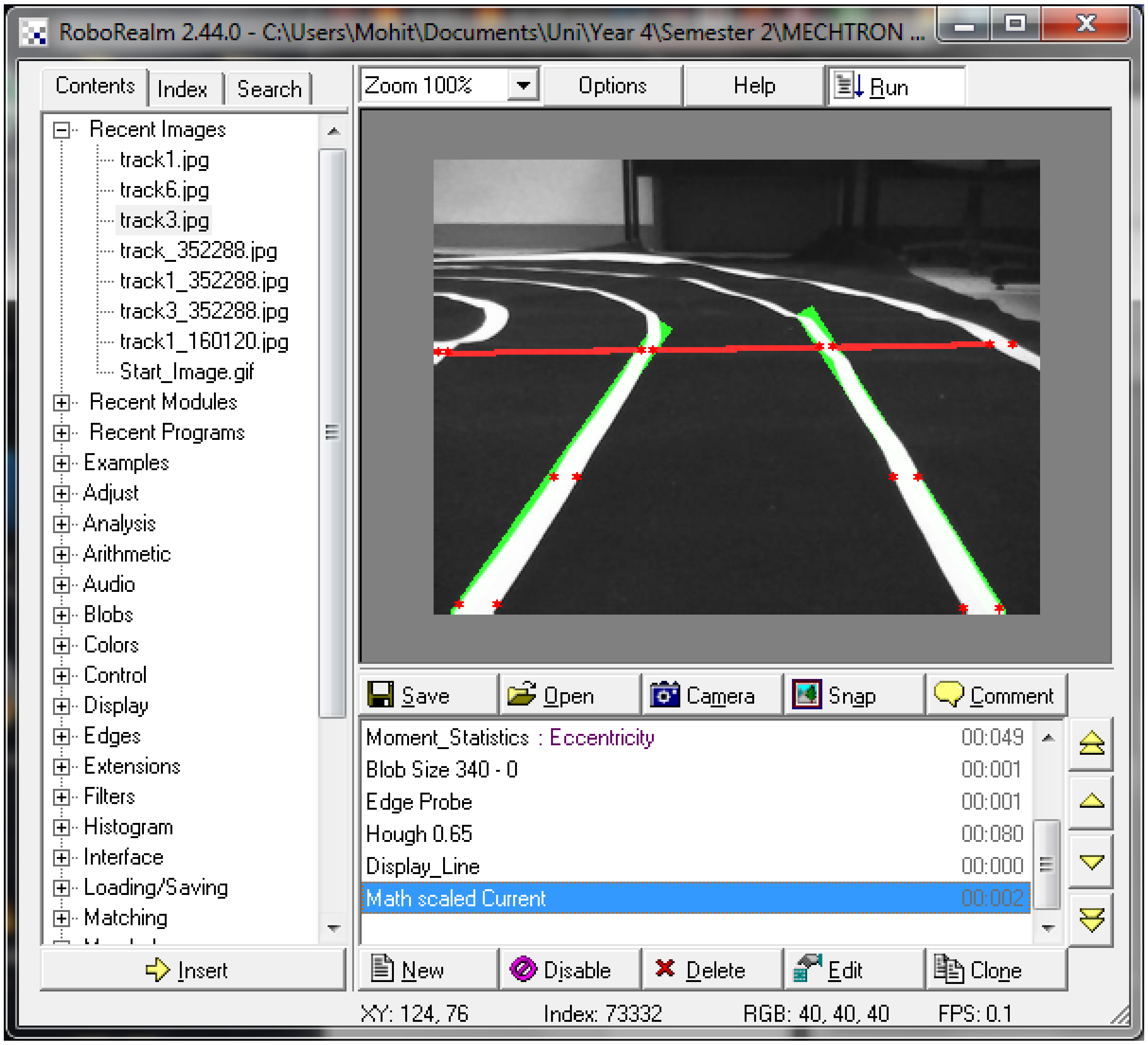
XY: 242, 104   Index: 100566   RGB: 0, 0, 0   FPS: 0.1



RoboRealm 2.44.0 - C:\Users\Mohit\Documents\Uni\Year 4\Semester 2\MECHTRON ...

```
Moment_Statistics : Eccentricity     00:050
Blob Size 340 - 0                    00:001
Edge Probe                           00:001
Hough 0.65                           00:080
Display_Line                         00:000
Math scaled Current                  00:002
```

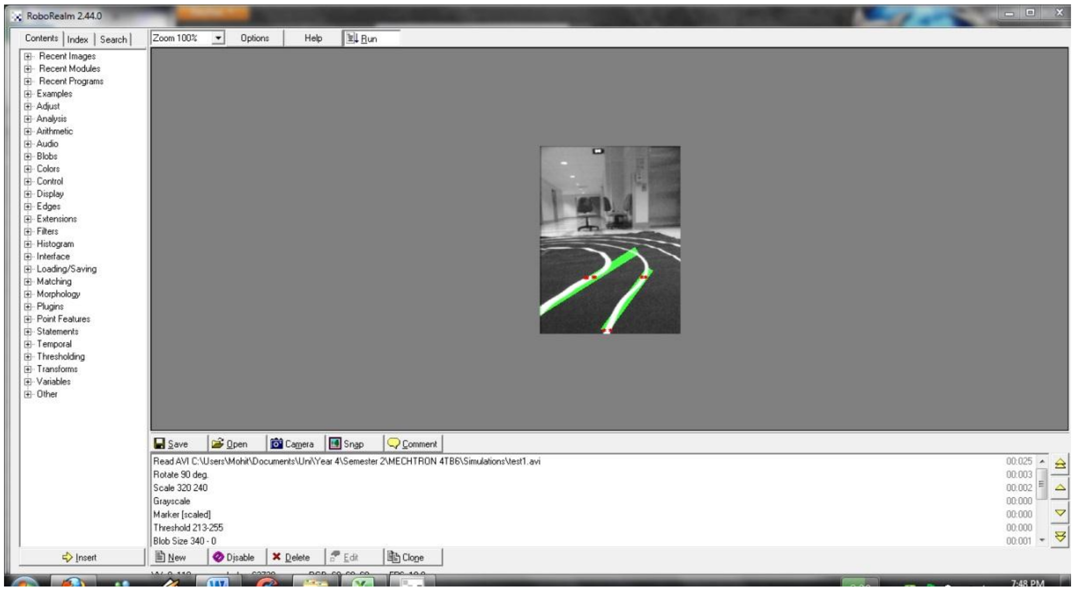
XY: 0, 239   Index: 229440   RGB: 0, 0, 0   FPS: 0.1



```
Moment_Statistics : Eccentricity   00:049
Blob Size 340 - 0                  00:001
Edge Probe                         00:001
Hough 0.65                         00:080
Display_Line                       00:000
Math scaled Current                00:002
```

XY: 124, 76   Index: 73332   RGB: 40, 40, 40   FPS: 0.1

After using Roborealm on a still image, a test video was taken using the camera intended to be used and the results were the following.



```
Read AVI C:\Users\Mohit\Documents\Uni\Year 4\Semester 2\MECHTRON 4TB6\Simulations\test1.avi   00:025
Rotate 90 deg.                                                                               00:003
Scale 320 240                                                                                00:002
Grayscale                                                                                    00:000
Marker [scaled]                                                                              00:000
Threshold 213-255                                                                            00:000
Blob Size 340 - 0                                                                            00:001
```

This is not an accurate representation as with our current hardware mounts the camera was not able to be placed in its normal orientation. This gives a good idea of the types of filters needed to be coded and what is possible to do in video processing in terms of frames per second. It was slowed down but this is done with no optimization in place.

In summary, more work needs to be done with the actual hardware to ensure proper performance to meet the requirements of this project.

6. Appendix A: Requirements

These requirements are presented here for any reader to easily compare the results to the requirements

	GOAL (not in any particular order)
1	KISS – Keep it Simple Stupid
2	Don’t crash
3	Stay within the boundaries of road
4	Stay completely within one lane when not changing lane
5	Drive forward when not avoiding obstacles
6	Emergency stop
7	Go as fast as possible without issues
8	Smooth motion of vehicle
9	Advertisement opportunities- money donated if profit
10	Follow other moving object
11	Look good
12	Stay within budget
13	Cost effective
14	Good code

Table 1: List of initial goals for project

1. KISS – Keep It Simple Stupid
    - Meet all requirements of the client, nothing less and nothing more
    - Keep design simple
  - Implement functionality in the most optimal space (hardware or software)

2. Don’t crash
    - Don’t make physical contact with other objects that are not the ground
    - Don’t cause the car to be able to lose physical control or continue driving when it has made physical contact with an object
  - Don’t cause the car to be able to lose electronic control

3. Stay within the boundaries of the road
  - The entire vehicle is within the boundaries of the road and no section of the car shall leave the boundary

4. Stay completely within one lane when not changing lane
    - The entire vehicle is within a single lane and no section of the car shall leave this boundary when not changing lanes
  - When it is decided to change the lane the vehicle is allowed to cross a single lane and afterwards the above is enforced

5. Drive forward when not avoiding obstacles
    - The vehicle will progress along the track with the front end facing forward as long as nothing is obstructing it's progress
    - When encountering an obstacle the vehicle may manoeuvre in alternate fashions without breaking any other rule
   - Do not react to an object that is not within a physically avoidable distance

6. Emergency stop
   - The car will have a way to come to a complete stop when encountering an object
   - An emergency stop will occur when no other options is available

7. Go as fast as possible without issues
   - Travel as fast as possible without breaking any other rules
   - Maximum speed should not put E-Stop in jeopardy

8. Smooth motion of vehicle
    - Acceleration
        - ➢ Maintain constant acceleration while accelerating
        - ➢ Do not exceed a predefined threshold of acceleration
    - Decelerate
        - ➢ Under normal conditions the vehicle should decelerate within a certain range to maintain smoothness. The vehicle should anticipate this
        - ➢ Under exceptional circumstances the vehicle may decelerate as quickly as possible
    - Turning
        - ➢ When turning the wheels will never skid
        - ➢ Navigate a turn at an appropriate and safe speed whilst maintaining smoothness

9. Advertisement opportunities – money donated if profit
   - TBD (Yet to be approved by Dr. Wassyng)
   - Sponsors logo placed onto the car with all profit donated to charitable cause

10. Follow other moving object
    - Not required but the ability to follow another car around the track will be a selling point.

11. Look good
    - Only the sensors of the vehicle should be visible outside the body (preferably)
    - The vehicle circuitry shall be as aesthetically pleasing as possible.
    - Only printed circuitry (No breadboards) optimized for space are allowed.
    - All hardware must be properly mounted and secured

12. Stay within budget
    - Final build must be =<750.
    - Budget does not include ‘test’ materials

13. Cost effective
    - No single aspect of the project shall consume the majority of the budget and time

- Whenever possible economic component shall be used
- Only one solution to one problem shall be used
- There shall be no unnecessary redundancy
- Minimize cost of time and money by avoiding unnecessary part orders

14. Good code
    - Each layer must be insulated from change.
    - Good variable/function naming scheme
    - Avoid major changes to code that's not your responsibility without co-operation
    - Use a consistent coding scheme throughout
    - All modules should only deal with their respective concern.
    - All values relating to real world principles will be dealt with in their real world units.
    - Standardize upon a programming language for each component
    - Only SI units shall be used throughout the project
    - Absolutely comment on WHY specific piece of code was used
    - Document all public-facing API of each module