Color Tracking Load Bearing Wheeled Rover

Erim Gokce, Alfonso Jarquin, Johnny Louis, Neha Chawla, Sabri Tosunoglu

Department of Mechanical and Materials Engineering
Florida International University
Miami, Florida 33174

egokc001@fiu.edu, ajraq004@fiu.edu, jloui034@fiu.edu, nchaw002@fiu.edu, tosun@fiu.edu

ABSTRACT
This paper entails the mechanisms of operation, prospective market desire and consumer benefit of a motorized autonomous rover with adaptable tracking and following capabilities.

Keywords
Wheeled rover, tracking, color tracking, load carrying.

1. INTRODUCTION
1.1. Problem Statement
As society continues to strive towards automation of repetitive or laborious tasks, it becomes necessary to locate the areas where simple implementation and energy-saving potential coincide. Manual labor is one such area, where significant scale of humans still perform physically demanding tasks that have the potential of being completely automated.

1.2. Motivation
There exists a multitude of benefits, both on the corporate and individual levels, in having the ability of an intelligent rover to follow a user-controlled moving target. The primary feature of such a device would be in towing of items from one location to the other. This rover can be used at industry for transporting tools or scrap materials or at home for grocery shopping. The main aim of this robot is to reduce the human effort in carrying heavy loads thus preventing any injury from lifting. It can also be used for the disbursement of pesticides across a pre-set grid or the chlorination of a swimming pool. In the interest of preserving the versatility of this rover, the concept build outlined herein will not contain a load-bearing or load-towing feature.

1.3. Literature Survey
Automated robots are currently used in several industrial applications. They are used in warehouses, factory, and to transport products or items from one point to another. These robots are normally traditional 4 wheeled moving platforms rovers. They are low to the ground with a very high load capacity. They can carry weight several times that of themselves. The research conducted prior to the design of the robot was carried out with the intent to learn not only about vehicle automation and drivability, but also the current uses, applications and models available to the public with similar uses.

Figure 1. Amazon warehouse automated guided vehicles (AGV)

The primary type of guided robot discovered in the research phase was an AGV, or automated guided vehicle. AGVs are platform like robots which follow predetermined paths from point to point and move cargo across busy warehouse floors. The AGVs use different markers to define their paths like laser triangulation, magnetic strips, colored lines, wire markers, magnetic grids, and natural feature recognition. Each type of technology has its own advantages and can be used according to the application of the robot.

2. DESIGN CRITERIA
We have used the colored line recognition and the natural feature recognition in our robot design. The colored line tracking will allow the robot to drive via an "eye" which is located on the front bottom side. The "eye" detects the colors on the floor and drives the robot while maintaining the colored line in sight at all times. Our design will be able to track users based on color and will also be programmed to maintain a certain distance between the robot and the user.

2.1. Selecting the Camera
An important feature of the rover would be its adaptability and versatility when it comes to following an objective. Since it is

necessary for the rover to stay locked onto a target, the need for the rover to feature a camera as its main tracking tool became apparent. Research was conducted into the use of pre-prepared libraries such as OpenCV, which when married with a camera, can perform as desired. We used PixyCam, an 80x80 pixel camera since it features the foundation of the data acquisition and relaying feedback mechanism that was required of this tracking system. The PixyCam can be programmed to track a specific hue of color, or a pattern of up to seven hues in proximity of each other. It is also capable of relaying coordinates of height, width, distance back to a microcontroller. This set of features would be integral in having the robot working properly

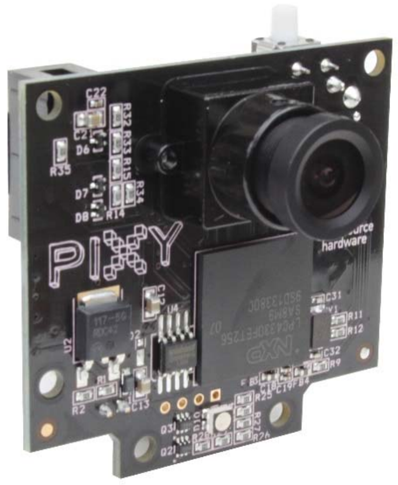


Figure 2. PixyCam, a crowd-funded tracking camera
.

2.2. Powering the System

The tracking rover would require a simple system of power that is long-lasting and low in weight. There are multiple components on the rover that require power, such as the camera, the microcontroller, and the motors. Since minimizing the weight is of the utmost importance, We chose to power the robot with a 9V dry cell battery and a 12V set of 8 AA batteries connected in-series. This method would prove to be optimal against other ideas such as a rechargeable battery pack, which could not supply the voltage required to drive the motors, as well as be the necessary fit and weight.

2.3. Driving the System

A set of DC motors that are used in various DIY RC home kits was selected for our prototype since it had a gear-ratio of 1:48 and a no-load speed of 200 RPM. While they are rated at 6V, we found that supplying twice the amount of voltage yielded no issues and with the inclusion of a motor shield on the microcontroller, which would be an Arduino Uno, the rover would be capable of being speed-controlled consistently with the included encoders.

For the small scale of this project, it was deemed best to favor DC motors over servomotors, for their power output. The Arduino Uno is incapable of supplying the sufficient wattage needed to drive both DC motors, so it became necessary to attach a separate motor shield, which would be independently powered by the 12V source. The 9V battery would power the Arduino, which connects through a ribbon cable to the PixyCam, powering it as well. This method supplies everything sufficiently and is low in weight and cost.

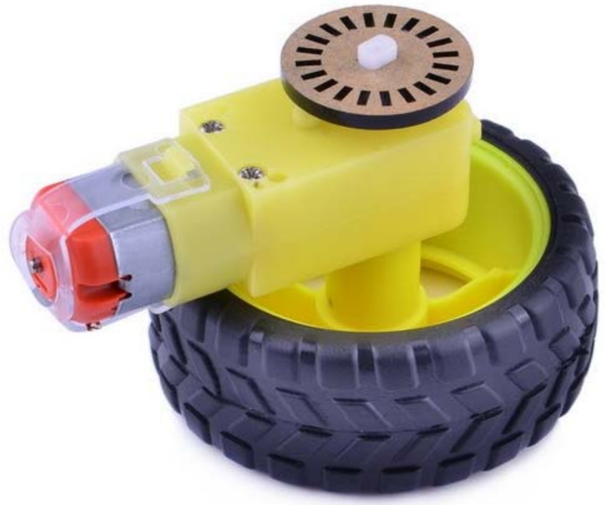


Figure 3. A DC motor with encoder and wheel attached

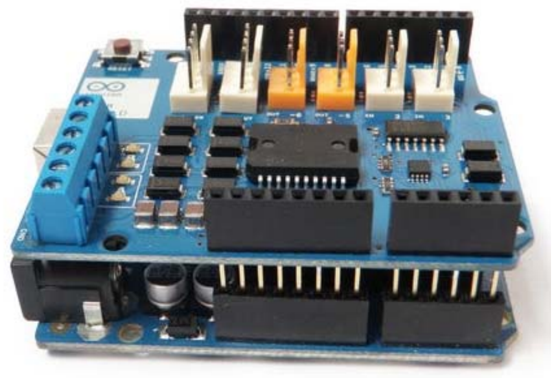


Figure 4. Motor shield mounted onto an Arduino Uno

For the included DC motor, there was also included a pair of encoders. These encoders can track the revolutions of the wheel in real time, and pre-setting a desired max RPM. Therefore, the supplied battery that is double the voltage that the motors are rated for should not be an issue. The encoders will keep the wheels spinning at the rate needed to keep up with the walking pace of an adult human, and this limitation of output will preserve battery life, effectively doubling the rate of runtime without needing to replace the batteries.

2.4. Rover Chassis

The chassis of the rover should be comprised of both metal and acrylic. The bottom base is metal, for stability and support, and the top level is acrylic, to house the electrical components since it will not conduct electricity.

2.5. 180° Rotation of View

The implementation of visuals in the robot must be capable of viewing angle of at least 180° to minimize power draw by preventing unnecessary rotation of the robot when possible, as well as provide more rapid and sophisticated tracking method. For this reason, a dual-servomotor setup was implemented into the base of the PixyCam, which would allow for both panning left to right and tilting up and down of the camera. The programming will notify the servos the correct time to rotate or tilt the PixyCam, per which motor is require to engage to keep the centroid of the object centered.

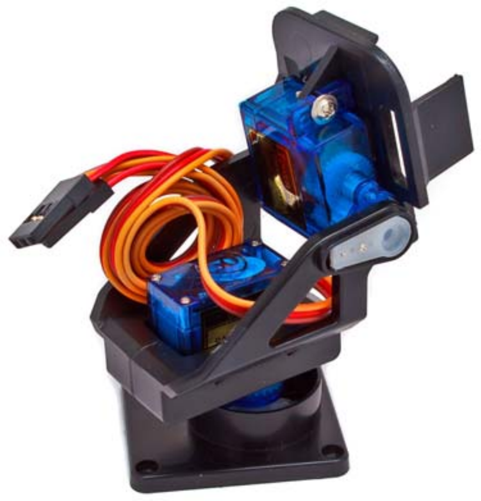


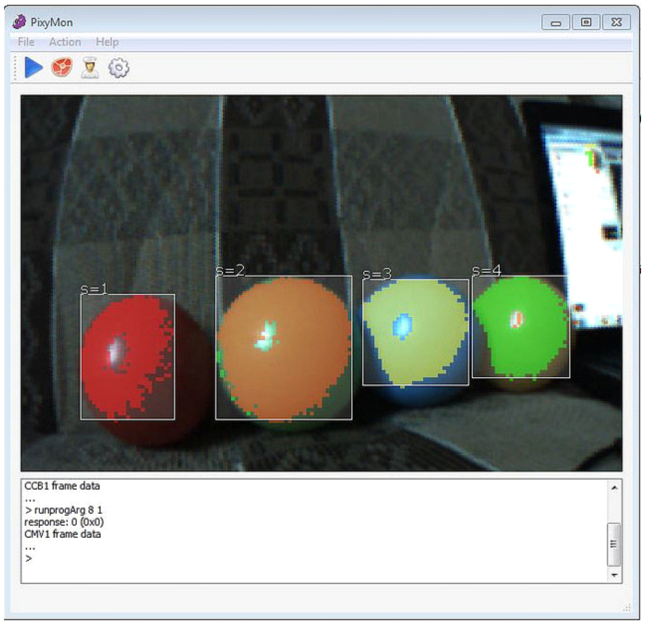
Figure 5. A pan/tilt servomotor setup, compatible with PixyCam

2.6. Communication and Instruction

For instruction of the PixyCam for when to pan and tilt, as well as for the communication to the motors of when and how rapidly to turn, the base hub of the system was chosen to be the Arduino Uno. The Arduino will be programmed with the instructions of how close to travel to the stimulus, obstacle avoidance, reversing, full rotation of the body, and full relaying of the coordinates provided from the PixyCam to the motor shield.

The methodology of tracking will be handled by the accompanying software, PixyMon. The software is intuitive to use and relays the necessary data. PixyMon allows tracking of up to seven different items, each item with the option of being comprised of just one tracked hue, or up to a pattern of seven. The tracked objects can be tracked to a much beyond needed distance, provided the object is of sufficient size. The PixyMon software separates each object into a "block" for which it assigns a signature. This block signature can be relayed back to the Arduino with important information, such as the height, width, and distance of the block. These parameters can be taken as X, Y, and Z coordinate data, and put to good use in telling the DC motors to operate. A computer is needed to tag blocks.

PixyMon is taught a block by holding up the item in front of the PixyCam, and selecting the "Set Signature" option. This allows the user to select in a rectangular box of desired size, the item shown on screen. PixyMon then highlights the entire object of the selected hue, and assigns it a signature. This is stored inside the PixyCam after disconnected from the computer.



```
CCB1 frame data
...
> runprogArg 8 1
response: 0 (0x0)
CMV1 frame data
...
>
```

Figure 6. The PixyMon tracking software interface

3. CONCEPTUAL PROTOTYPE DESIGN

The initial, conceptual design of the robot was brainstormed to be constructed out of 3D printed ABS plastic. The base would be a printed piece in the shape of a skateboard, and attached to stacked platforms inside which electronics and batteries could be stored. The design features a small compartment on the back to carry items, and the propulsion would be through front-wheel drive. A rear idler wheel would keep the platform stable.

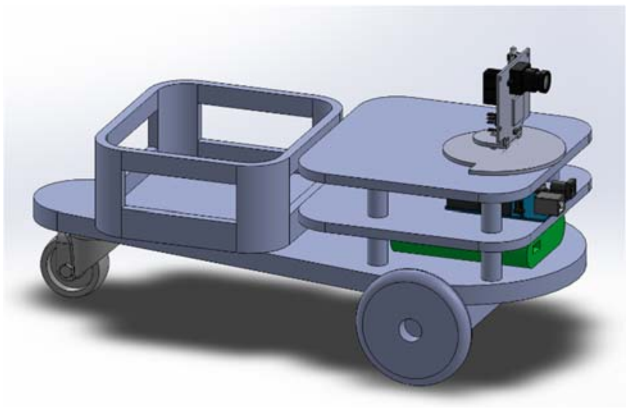


Figure 7. A conceptual design of the robot, with storage crate

We discovered that the method of 3D printing was more cost-prohibitive than purchasing multiple pre-made platforms which could be adjusted to affix to one another. Also, a design choice was made to remove the storage crate from the final prototype, to preserve versatility. It was decided to instead run a side production of a towing crate which could be affixed to the final robot, and could be reliant on its power output to be carried around.

An additional feature of the conceptual design was that it would be run off a rechargeable battery pack (See Fig. 7, battery pack in green). This was removed in the final product to minimize weight.

4. FINAL PROTOTYPE DESIGN

The completed prototype is constructed from an aluminum base which houses the wheels, batteries, and DC motors. The base connects to an acrylic roof, on top of which the electrical components are mounted. The Arduino Uno, motor shield, and PixyCam are secured to the top through holes drilled into the acrylic. These components connect through wiring to the bottom metal base. The wheels attach to the DC motors, those of which are secured to the metallic bottom through hot glue. The front idler wheel was screwed in to the slots in the front of the chassis.

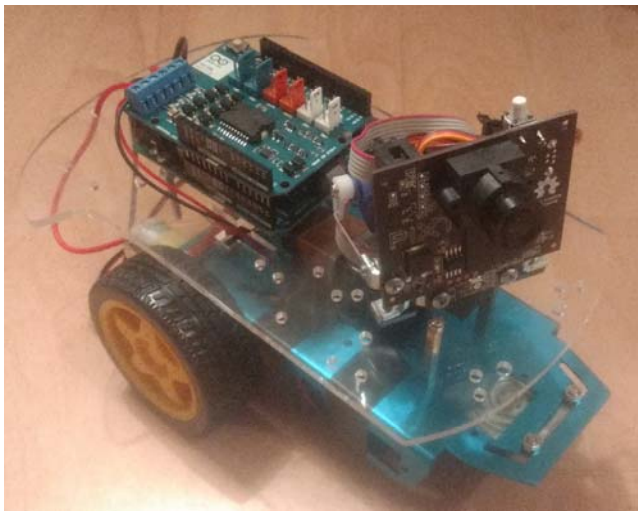


Figure 8. The completed prototype

As mentioned prior, the prototype does not feature a load bearing capacity. However, the prototype can withstand towing a load of 1-2 lbs, which can be stowed on a platform that can be easily attachable to a hitch on the rear of the robot. A detriment of including a hitched tow would be in the reversing of the robot, so it is of our's interest to gauge the demand for this product and, if sufficiently significant, modify the design to include the load crate.

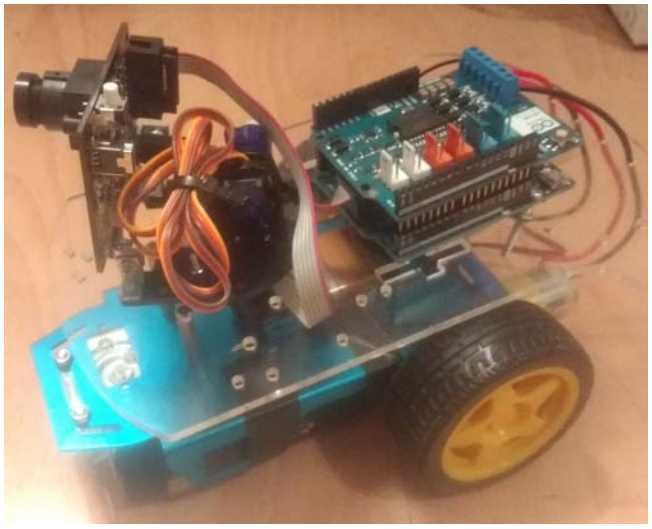


Figure 9. A side view of the robot

Certain design choices were made that vary from the original concept. The final robot is now rear-wheel driven, as opposed to front wheel driven. The multi-level platforms were removed in favor of one singular platform, which would improve stability. The batteries are mounted on both the top and bottom of the metal portion, with the bottom battery pack secured with Velcro. The wheels are wider than in the original concept. There is sufficient clearance between the batteries and the ground.

4.1. Size and Weight Specifications

The robot weighs just over eighteen ounces. It is eight-and-a-half inches long and six inches wide. The height is six inches. The robot was designed to be scalable, so a consumer-ready product would scale to two feet and ten inches long, two feet wide, and two feet high. The final product would be much denser, approximately 20 pounds due to the batteries, all-metallic infrastructure, and powerful motors.

4.2. Cost Breakdown Analysis

Much of the expenditure was towards the PixyCam. However, it is of important note that this price will remain stagnant and will not become costlier with the full scaled model. Many of the components were already on-hand, however we felt it necessary to identify the true cost of this project.

Table 1. Total Cost Breakdown of Color Tracker Robot

Component	Quantity	Price
Arduino Uno	1	$19
Metallic Platform	1	$16
Motor Shield	1	$25
Acrylic Platform	1	$12
Wheels	2	$4
9V Battery	1	$2
AA Battery	8	$5
PixyCam	1	$79
Servos for Pixy	2	$20
DC Motors	2	$10
Cables/Connectors	Various	$10
FINAL		**$202**

4.3. Methodology of Connection

The 12V battery pack supplies its energy through a positive and negative cable, which secures into the motor shield in the SUPPLY and GND terminals. This motor shield has two output nodes which route the voltage into the two DC motors through positive and negative wire which are soldered onto the motor terminals. The motor shield rests directly on top of the Arduino, and is secured through the numerous pins which are custom-matched to fit into the Arduino's analog, digital, and power inputs. The Arduino itself is powered through the 9V cable. The PixyCam shares power draw with the Arduino, and is connected to it through a custom ribbon cable attached to an in-circuit serial programmer node. Finally, the PixyCam is secured to a base which we constructed with two 180° servomotors, which control the pan and tilt functions of the PixyCam.

5. CONTROL THEORY

The bulk of the control theory is through the relaying of information from PixyCam to the motors. It was necessary to program the Arduino to be the middleman in this interaction.

5.1. Language and Method

The language used was C++. The code required an object-oriented programming language, and C++ is native to the Arduino environment. The code relies heavily on object calls and if statements. The idea is to teach PixyCam an object, and scrape the data it relays into Arduino, which is taught to react to that information accordingly.

5.2. Code Breakdown

The code begins by establishing a servo loop class, a proportional/derivative feedback loop for the pan and tilt servo tracking of the "block". This ensures that the robot remains stationary if the angle is changing, but the proximity of block to robot is unchanged.

```
class ServoLoop
{
public:
ServoLoop(int32_t proportionalGain, int32_t
derivativeGain);
void update(int32_t error);
int32_t m_pos;
int32_t m_prevError;
int32_t m_proportionalGain;
int32_t m_derivativeGain;};
// ServoLoop Constructor
ServoLoop::ServoLoop(int32_t proportionalGain,
int32_t derivativeGain)
{
m_pos = RCS_CENTER_POS;
m_proportionalGain = proportionalGain;
m_derivativeGain = derivativeGain;
m_prevError = 0x80000000L;}
```

The servo loop then updates, by calculating a new output based on the measured error and the current state.

```
void ServoLoop::update(int32_t error)
{
long int velocity;
char buf[32];
if (m_prevError!=0x80000000)
{
velocity = (error*m_proportionalGain + (error -
m_prevError)*m_derivativeGain)>>10;
m_pos += velocity;
if (m_pos>RCS_MAX_POS)
{
m_pos = RCS_MAX_POS;
}
else if (m_pos<RCS_MIN_POS)
{
m_pos = RCS_MIN_POS;
}
}
m_prevError = error;}
```

Next was the creation of a simple motor class. This would communicate to the motor driver, which utilizes timer 1 on the Arduino, to control the DC motors using a 20-kilohertz pulse-width modulation.

```
class SimpleMotors
{
public:
// constructor (doesn't do anything)
SimpleMotors();
// enable/disable flipping of motors
static void flipLeftMotor(boolean flip);
static void flipRightMotor(boolean flip);
// set speed for left, right, or both motors
static void setLeftSpeed(int speed);
static void setRightSpeed(int speed);
static void setSpeeds(int leftSpeed, int rightSpeed);
private:
static inline void init()
{ static boolean initialized = false;
if (!initialized)
{ initialized = true;
init2();
}
}
// initializes timer1 for proper PWM generation
static void init2();
};
#define PWM_L 10
#define PWM_R 9
#define DIR_L 13
#define DIR_R 12
#if defined(__AVR_ATmega168__) ||
defined(__AVR_ATmega328P__) || defined
(__AVR_ATmega32U4__)
#define USE_20KHZ_PWM
#endif
static boolean flipLeft = false;
static boolean flipRight = false;
```

After the simple motor class was established, it was important to create the reaction of the robot car to orient itself to face the tracked block one PixyCam is normalized. This occurs once the block begins changing proximity to the robot. Size is the area of the object, and a running average of eight sizes are kept.

```
int32_t size = 400;
void FollowBlock(int trackedBlock)
{
int32_t followError = RCS_CENTER_POS -
panLoop.m_pos;
size += pixy.blocks[trackedBlock].width *
pixy.blocks[trackedBlock].height;
size -= size >> 3;
int forwardSpeed = constrain(400 - (size/256), -100,
400);
int32_t differential = (followError + (followError *
forwardSpeed))>>8;
int leftSpeed = constrain(forwardSpeed + differential,
-400, 400);
int rightSpeed = constrain(forwardSpeed - differential,
-400, 400);
```

Forward speed will decrease as the robot approaches the object. The steering differential is proportional to the error times the forward speed, and this adjusts the left and right speeds.

```
if (k%50 ==0 ){
Serial.println("Speeds in follow");
Serial.println(rightSpeed);
Serial.println(leftSpeed);
}
motors.setLeftSpeed(leftSpeed);
motors.setRightSpeed(rightSpeed);
}
```

Another important prompt was to instruct the PixyCam to search for blocks, by panning back and forth at random until a block with the "correct" hue is detected, after which the following begins.

```
int scanIncrement = (RCS_MAX_POS -
RCS_MIN_POS) / 150;
uint32_t lastMove = 0;
void ScanForBlocks()
{
if (millis() - lastMove > 20)
{
lastMove = millis();
panLoop.m_pos += scanIncrement;
if ((panLoop.m_pos >=
RCS_MAX_POS)||(panLoop.m_pos <=
RCS_MIN_POS))
{
tiltLoop.m_pos = random(RCS_MAX_POS * 0.6,
RCS_MAX_POS);
scanIncrement = -scanIncrement;
if (scanIncrement < 0)
{
motors.setLeftSpeed(-250);
motors.setRightSpeed(250);
}
else
{
motors.setLeftSpeed(+180);
motors.setRightSpeed(-180);
}
delay(random(250, 500));
}
pixy.setServos(panLoop.m_pos, tiltLoop.m_pos);
}
}
```

The robot is intelligent enough to reverse when it is being approached by the tracked object, and to not begin movement until the tracked item changes in proximity to the robot. The distance is measured by multiplying the height and width variables that the PixyCam can measure.

```
void setup()
{Serial.begin(9600);
Serial.print("Starting...\n");
motors.setSpeeds(0, 0);
pixy.init(); }
uint32_t lastBlockTime = 0;
```

The main loop is included below. This runs continuously after setup, which precedes the main loop and runs at startup.

```
void loop()
{
if (k%50 ==0 )
Serial.println("Loop Started");
k++;
uint16_t blocks;
blocks = pixy.getBlocks();
// If we have blocks in sight, track and follow them
if (blocks)
{
if (k%50 ==0 )
Serial.println("blocks in sight");
int trackedBlock = TrackBlock(blocks);
FollowBlock(trackedBlock);
lastBlockTime = millis();
}
else if (millis() - lastBlockTime > 100)
{
motors.setLeftSpeed(0);
motors.setRightSpeed(0);
ScanForBlocks();
}
}
int oldX, oldY, oldSignature;
```

6. EXPERIMENTATION

The robot was tested by setting it to track a red hat that the tester would hold in front of it. The lighting conditions of the room were optimal, with no natural daylight interference. The PixyCam is not as functional when daylight is factored in, and as such, has poorer outdoor performance. Nonetheless, the robot was tested both inside at nighttime and outside with daylight.

7. RESULTS

Testing was successful in the indoor test, and the robot tracked and followed the object for five minutes. The robot lost visual sight of the hat once, due to the lighting conditions of the testing room, but the panning back-and-forth code block restored the visual sight to the hat, and the robot could continue following the tester. The robot is calculated and expected to last up to thirty minutes of continuous motion following. It was able to achieve a maximum speed of four miles per hour.

The outdoor test proved difficult, with the red hat reflecting much of the light and causing PixyCam to behave erratically. We were able to remedy this issue, by programming the PixyCam with a custom-made color signature, which comprised of a piece of paper with three different unique and bright colors. The robot functioned normally until subjected to direct contact with sunlight, which interfered in the tracking. PixyCam should be avoided from direct sunlight.

PixyCam's lens is focused through screwing in/out, and as such it is recommended to users to secure the lens in place once it is set. We encountered multiple scenarios of the camera losing focus due to a loose lens that was turning on its own due to the motion of the moving robot.

8. CONCLUSION

The concept of tracking a pattern by a platform has been demonstrated in terms of a first prototype and encouraging results have been observed. Application of this concept in industrial as well as personal robotics offers a potentially wide spectrum of new designs for robotic platforms and even manipulators.

In conclusion, we successfully built the color tracking rover which would identify the programmed color and then follow the path taken by the color. Further research can be carried out on programming the rover to move irrespective of the light conditions.

9. ACKNOWLEDGEMENTS

We would like to acknowledge Gene Yllanes for her assistance in construction of the programming code.

10. REFERENCES

[1] Bhasin, Kim, and Patrick Clark. "How Amazon Triggered a Robot Arms Race." *Bloomberg Techology*. Bloomberg, 16 June 2016. Web. 27 Apr. 2017.

[2] "CMUcam5 Pixy." *Wiki - CMUcam5 Pixy - CMUcam: Open Source Programmable Embedded Color Vision Sensors,* 2017.

[3] "Arduino UNO Rev3." Arduino Store USA. N.p., n.d. Web. 27 Feb. 2017.

[4] Cade, D. L. "Pixy: A Low Cost Camera that Recognizes and Follows bjects by Color." *PetaPixel*. N.p., 01 Sept. 2013. Web. 27 Apr. 2017.

[5] Knight, Will. "An Italian scooter maker invents a robot that follows you around carrying your stuff." *MIT Technology Review*. MIT Technology Review, 07 Feb. 2017. Web. 27 Apr. 2017.

[6] "Adafruit Motor Shield V2 for Arduino." *Using DC Motors | Adafruit Motor Shield V2 for Arduino, Adafruit Learning System*. N.p., n.d. Web. 27 Apr. 2017.

[7] *Product Details and Description*. N.p.: n.p., n.d. *Mantech*. Feetech, 2013. Web. 27 Apr. 2017.

[8] Team, Editorial. "Control a DC Motor with an Arduino." *All About Circuits*. N.p., 07 July 2015. Web. 27 Apr. 2017.

[9] "Tutorial: Pixy (CMUcam5)." *Physical Computing*. IDeate, n.d. Web. 27 Apr. 2017.