Arduino and Android Powered Object Tracking Robot

Mejdl Safran and Steven Haar
Department of Computer Science
Southern Illinois University Carbondale
Carbondale, Illinois 62901 USA
mejdl.safran@siu.edu, info@stevehaar.com

Abstract

We have built a four wheeled robot with an Arduino microcontroller, specifically the Arduino Mega 2650. We have written Arduino and Android libraries to allow an Android device to control the robot through a USB connection. The robot is designed to track objects by spinning left and right to keep the object in sight and driving forward and backward to maintain a constant distance between the robot and the object. Images are acquired through the camera of an Android device which is attached to the robot. The camera is attached to servos on the robot which allow the camera to pan and tilt. Several image processing techniques are used to detect the location of the object being tracked in the images.

Two different kernel based trackers are implemented as Android applications. One of them uses a color based tracking method and the other uses a template based tracking method. Both applications use Android's OpenCV library to help with the image processing. The experimental results of the robot using both methods show robust tracking of a variety of objects undergoing significant appearance changes, with a low computational complexity.

I. INTRODUCTION

We have built a mobile robotic system for tracking and following moving objects. Such a system provides important capabilities for assistance of humans in various settings, e.g., home use, health care and transportation. Tracking with mobile robots is an active research area and many successful systems have been developed, such as hospital assistance [5] and pedestrian tracking [6]. The reason for using a mobile robot is that a mobile robot can cover a wide area over time and can reposition itself in response to the movement of the tracked objects for efficient tracking.

Tracking is not only important for mobile robotic systems but also for a number of applications. Visual surveillance, motion capture and medical imaging all require robust tracking of objects in real time. The task of tracking becomes difficult to handle when the target objects change their appearance and shading conditions. Moreover, an important parameter of the tracker is the computational complexity which determines whether the tracker can be used in real time applications or not. The tracking methods used by the robot described in this paper overcome these main challenges.

To track an object, two Android applications which use image processing technology were implemented. The applications were installed on an Android device which was attached to a four wheeled robot powered by an Arduino microcontroller. The Android device is used to control the robot through the Arduino microcontroller and process the images acquired through its camera. All image processing tasks are performed using the Android device without the need of sending the images to a server to perform the image processing tasks.

One of the Android applications uses color based tracking method and the other uses a template based tracking method. The color based tracking method uses range thresholding and contour detection techniques. The template based tracking method uses Powell’s direct set method for object localization [2]. Both applications use Android's OpenCV library [1] to help with the image processing.

The experimental results of the robot using the color based tracking method show robust tracking of colored objects at an average frame rate of 25 frames per second, which is sufficient for real-time applications. The template based tracking method is not so efficient for real-time applications due to the less powerful processor in Android devices. However, the experimental results of the template based tracking method using a more powerful processor and a stationary camera show efficient tracking of any object regardless the color and shape at an average frame rate of 80 frames per second, which is sufficient for real-time applications. All experimenters show that our robot is a low cost, high performance robot.

The rest of the paper is organized as follows. Section II reviews related work. Section III describes our robot and its components. Section IV shows how an android device controls the robot. Section V describes the two methods used for tracking. Section VI shows our experimental results, and finally Section VII concludes our work.

II. RELATED WORK

Our project uses the Android Open Accessory Development Kit for communication between the Android device and the Arduino microcontroller. The Arduino microcontroller can sense the environment by receiving input from the Android device and can affect its surroundings by controlling the motors and the Android device's camera attached to the robot. We modified and expanded the concepts and the sample code in [7] to establish the communication link between the Android device and the Arduino microcontroller.

A number of tracking algorithms have been proposed in the literature. The trackers presented in this paper and used in our project are kernel based trackers that have gained popularity due to their simplicity and robustness to track a variety of objects in real time. Kernel based trackers can be classified [9] into three main classes. (1) template trackers; (2) density-based appearance model trackers; (3) multi-view appearance model trackers. Our project uses the template tracker and the density-based tracker.

Two different kernel based trackers are implemented. The first is color based tracking method. It uses range thresholding and contours detection techniques which are basic concepts in the field of digital image processing [8]. The second is template based tracking method that uses Powell's direct set method for object localization [2]. The Android's OpenCV library is used to help with the image processing for both trackers. In our project, we did not include obstacle avoidance or occlusion handling. These two issues are left as a future work.

III. ROBOT

The robot consists of the following components that are listed with their online links in Appendix A. This section describes each component in detail.

1. Lynxmotion - A4WD1 Chassis
2. Lynxmotion - Off Road Robot Tires
3. Arduino - Arduino ADK R3
4. Pololu - 50:1 Metal Gearmotor 37Dx54L mm with 64 CPR Encoder
5. Sabertooth - Dual 12A Motor Driver
6. MaxBotix - MB1000 LV-MaxSonar EZ0
7. Lynxmotion - Base Rotate Kit
8. Lynxmotion - Pan and Tilt Kit
9. Hitec Robotics - S-422 Servo Motor
10. Tenergy Coporation - Li-Ion 18650 11.1V 10400mAh Rechargeable Battery Pack

To give us the best flexibility, we built our robot from scratch rather than purchasing a robot kit. The chassis [A1] consists of an 8" x 9.75" aluminum base with flat lexan panels on top and bottom. For the wheels we used large 4.75" diameter off road rubber tires [A2]. The Arduino microcontroller is mounted inside the robot. We chose to go with the Arduino Mega 2560 ADK board [A3]. We chose this board because it is specifically designed to be used with the Android Open Accessory Kit. The board has a USB A plugin for plugging in an Android device. Figure 1 shows the Arduino Mega 2560 ADK board.

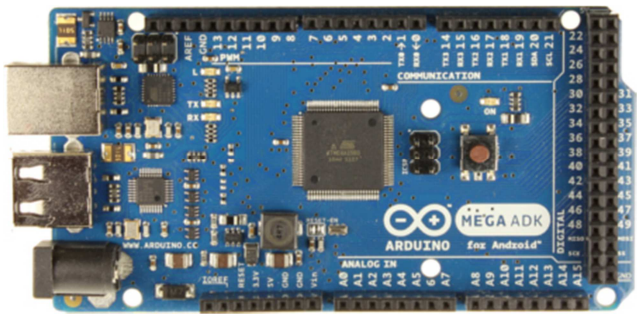


Figure 1: shows the Arduino Mega 2560 ADK board

To power the wheels we purchased four 12v 50:1 gear motors [A4]. The gear motors run at 200 rpm and have a stall torque of 12 kg-cm. Each motor also has a built in encoder to provide feedback to the microcontroller on how far the wheel has turned. Because the Arduino microcontroller is not capable of powering such 12v motors, we used a 12 amp dual motor driver [A5]. This motor controller allows the wheels to be controlled in two independent sets. In order to steer the robot left and right we connected both left wheels to one channel and both right wheels to the other channel. Figure 2 shows the gear motors and motors controller used in our project.

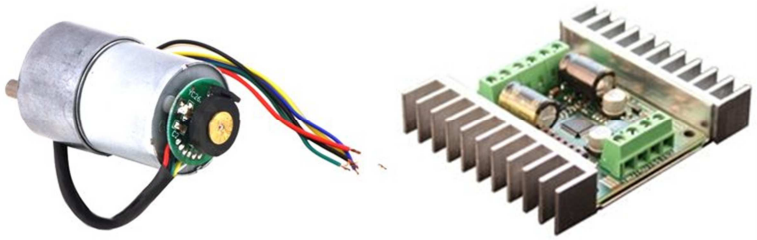


Figure 2: gear motors and motor controller

To enable the robot to detect obstacles while driving forward or backward, we installed an ultrasonic rangefinder [A6] in the front and rear of the robot, Figure 3. Because we wanted to detect large obstacles which the robot could not navigate over, we chose to use a rangefinder with a wide beam pattern.

Figure 3: ultrasonic rangefinder

To give the Android device range of motion, we installed a rotating base [A7] on top of the robot. This allows the Android device to rotate 180 degrees horizontally. On top of the rotating base we attached a pan and tilt kit [A8] which held the dock for the Android device. This kit allows the Android device to tilt up and down. To power the rotating base and the pan and tilt kit, we used several small servo motors [A9]. Figure 4 shows the rotating base, the pan and tilt kit and the servo motor, respectively.

Figure 4: rotating base, pan and tilt kit and the servo motor

As for the power supply we used an 11.1V 10.4 amp lithium ion battery [A10]. We employed a series of splitters and switches to connect power from the battery to the microcontroller and to the motor driver. Figure 5 shows the robot after installing all the components. For more pictures of the robot, see Appendix B.

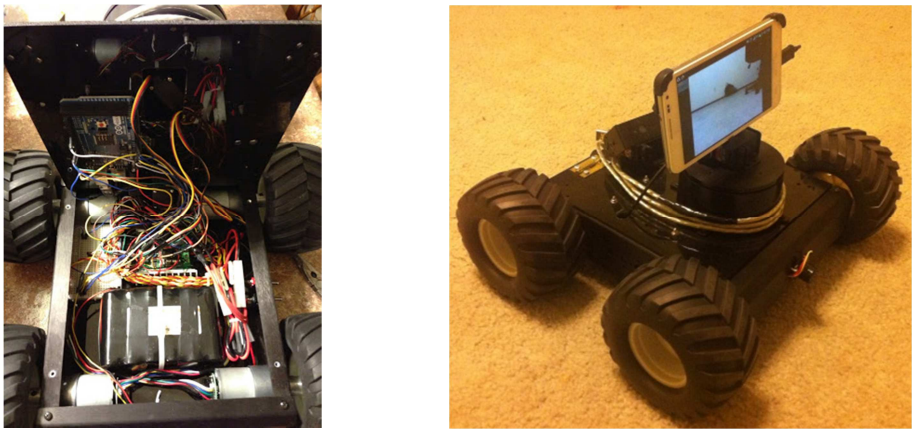


Figure 5: The robot after installing its components.

IV. Android

The Android device plugs into the microcontroller with a standard USB A to USB mini cable. We created two Android libraries that are used by our Android app. The video processing library retrieves raw data from the Android's camera and passes this data to a frame processor one frame at a time. It is the job of the frame processor to analyze the data and detect the location of the object in the given frame. The frame processor then invokes a method in the main Android application which in turn uses the robot controller library to control the robot.

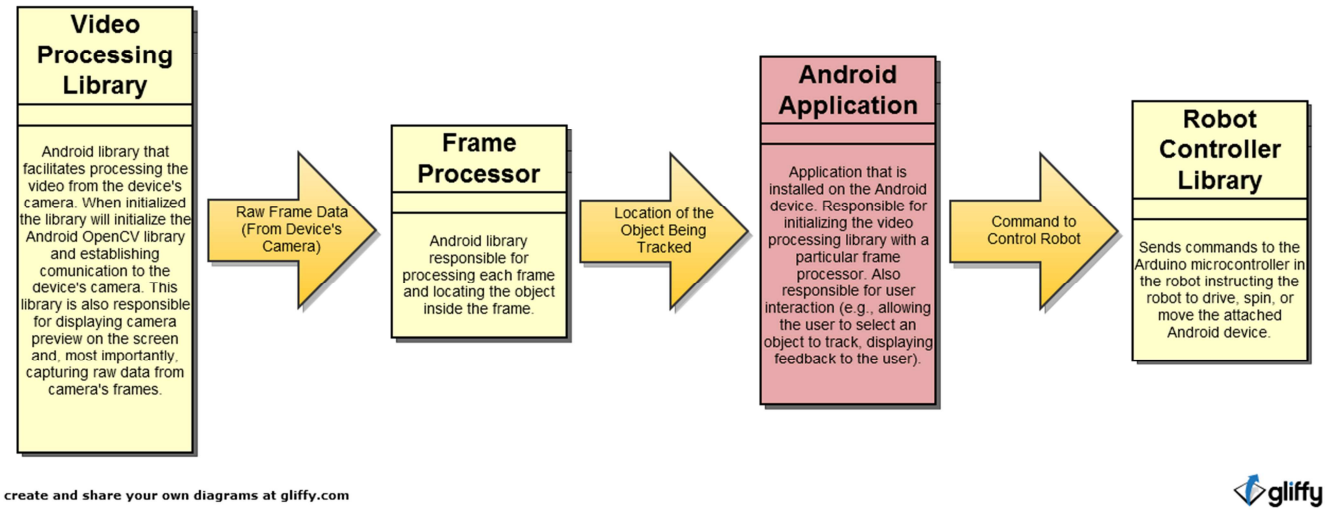


Figure 6: Communication between the implemented libraries

A. Video Processing Library

The video processing library is an Android library contains several classes for processing frames from the Android's camera and passing the frame data to the frame processor. When initialized, this library will initialize the Android OpenCV library and establish communication to the Android device's camera. OpenCV is an open source image processing library available for Android as well as many other platforms. In order to use the OpenCV Android library in any part of an application, the library must be properly initialized. Initializing the library is just "boiler plate" code and many frame processors will use the OpenCV library. It is for these reasons that the video processing library takes it upon itself to initialize the Android OpenCV library. The video processing library accesses the camera on the Android device and captures frames one at a time. The frames are then sent to a frame processor for processing.

B. Frame Processor

The frame processor attempts to detect where the object is located in the frame and then invokes a method in the main android application, passing in information obtained from processing the camera frame. In this way multiple different frame processors can be written and can be swapped in and out to be used with different Android applications. We have implemented two frame processors that we have included in the library. Both of these frame processors are discussed in detail in the Section V.

C. Android Application

This refers to the main application that is installed on the Android device. This application will use the other libraries mentioned. This application is responsible for initializing the video processing library with a particular frame processor. Also it is responsible for all user interaction (e.g., allowing the user to select an object to track, displaying feedback to the user).

D. Robot Controller Library

The robot controller library is an Android library which provides an object oriented interface for controlling the robot from within an Android application. The library works by establishing a USB connection between the Android and the Arduino microcontroller. Once the connection is established, the library exposes a set of methods that can be called from any Android code through a Robot class. The seven primary methods exposed by the Robot class are explained below.

1) **Drive Forward / Backward**

   This method accepts a speed (0 – 100) as an argument. This tells the robot to drive forward or backward indefinitely. The robot will only stop if it detects that it will collide with an obstacle or if it receives another command.

2) **Travel Forward / Backward Distance**

   This method accepts a speed (0 – 100) and a distance (in centimeters) as arguments. The robot will drive forward or backward, but will use its drive motor encoders to stop once it has travelled the specified distance. The robot will also stop if it detects that it will collide with an obstacle or if it receives another command.

3) **Spin Left / Right**

   This method accepts a speed (0 – 100) as an argument. This tells the robot to spin left or right indefinitely. In order to do this, one set of wheels (either left or right) rotates forward while the other set of wheels rotates backward.

4) **Rotate Left / Right**

   This method accepts a speed (0 – 100) and a degree (e.g., 45°, 180°, 360°, 720°) as arguments. The robot will spin left or right, but it will use its drive motor encoders to stop once the robot has rotated according to the specified degree.

5) **Rotate Android Horizontally**

This method accepts a degree (0 to 180) as an argument. This will rotate the base the Android rests on.

6) **Tilt Android Vertically**

This method accepts a degree (0 to 180) as an argument. This will turn the servo holding the Android dock which will cause the Android to tilt up or down.

7) **Track Object**

This method accepts a rectangle as an argument. This method should be called once for each frame processed. The rectangle should represent the coordinates of the object being tracked in the processed frame. This method works by analyzing the rectangle passed in and comparing it to the center of the frame and to an original object rectangle. The robot attempts to keep the object centered in the frame. By comparing the rectangle to the center of the frame the robot can determine if should tilt or rotate the Android device or if it should spin its wheels in an effort to center the object. By comparing the rectangle to the original object rectangle, the robot can determine if the object is getting bigger or smaller, (i.e. if the object is moving closer or further away). This will dictate whether the robot will drive forward, backward, or remain stationary. When tracking an object, the robot maintains an internal state. Each time the track object method is called (once per processed frame) the robot acts according to its current state and regarding the rectangle passed into the method.

When driving forward and backward the robot uses a proportional control method to keep the robot driving in a straight line. One side of the robot (either left or right) is marked as the master and the other side is marked as the slave. The robot uses its drive motor encoders to detect the rotations of the master and slave sides of the robot. Periodically, the robot will poll the encoders and make adjustments to the speed of the motors. The slave side motors remain unchanged, but the master side motors are adjusted in an effort to keep the robot traveling on a straight line. For example, if the master side motors have rotated more than the slave

side motors, then the master side motors are slowed down. If the master side motors had rotated less than the slave side motors, then the master side motors would have their speed increased.

In addition to calling these methods, an Android application can also add a listener to the robot and will be notified periodically of the status of the rangefinder. The rangefinder statuses are given in centimeters which represents the distance between the sensor and a known obstacle. The following diagram, Figure 7, is the state diagram of the robot.

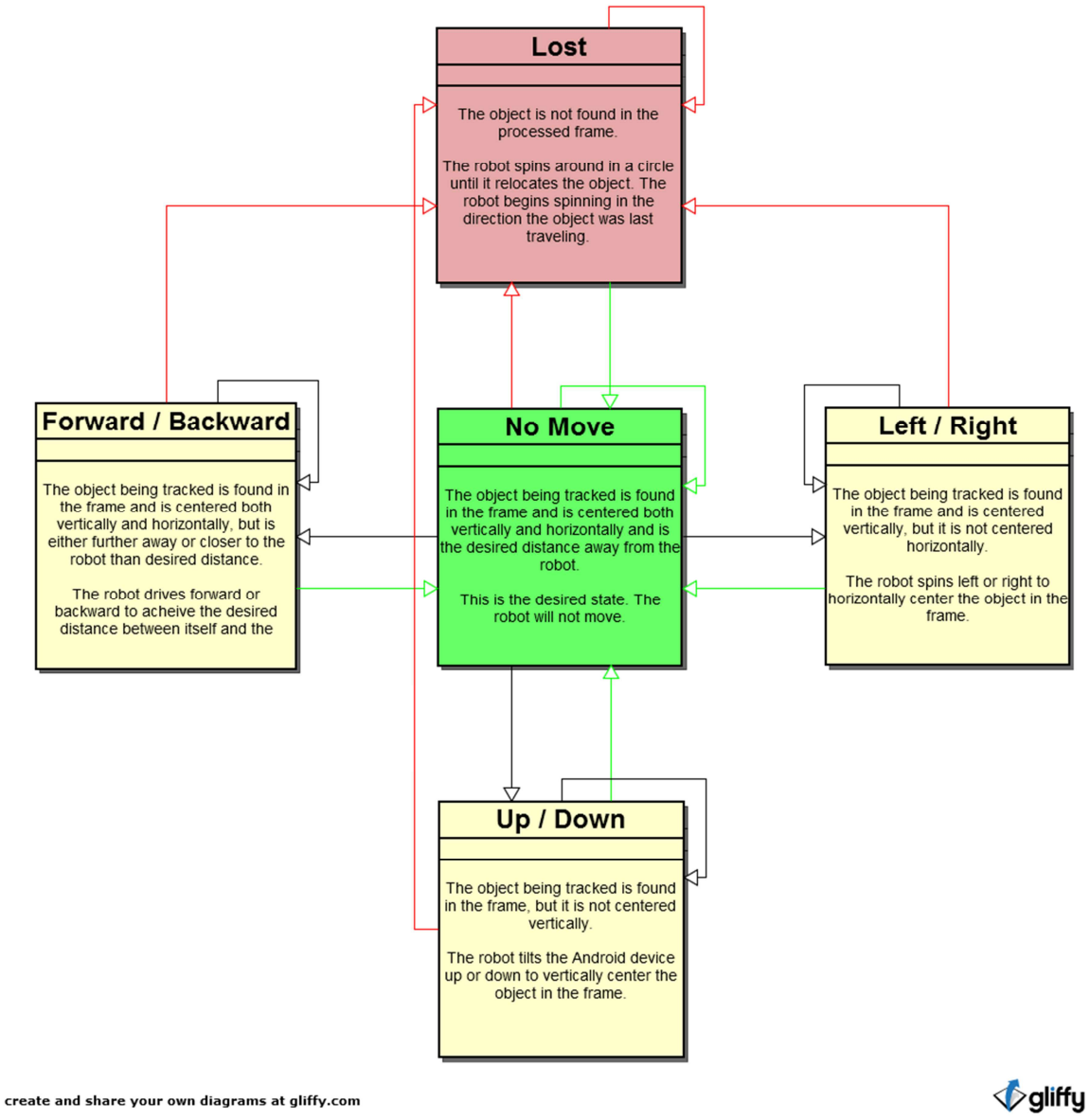


Figure 7: The state diagram of the robot

V. TRACKING

When the android device is ready to process the frames received from the camera, the task now is how the frames are processed to identify the new position of the tracked object. In our project we used two different methods for object tracking: color based tracking and template based tracking. Any new methods can be easily added to the system since the system is built to be extendable. Both of these methods will be discussed in detail in this section.

A. Color Based Tracking Method

The method starts from the first frame when the user touches the object that is wanted to be tracked on the android device screen. The result of the user touch is an RGB pixel. Using only one pixel to determine the target color is not sufficient. So we define a touched rectangle by including four neighbor pixels from each side, i.e., forming 9 x9 rectangle. Then, the touched region will be converted from RGB color space to HSV color space. After producing the HSV touched region, the average of each component (hue, saturation and value) is computed among all the pixels in the touched region, called average touched pixel.

Looking for the exact values of the average touched pixel in the next frame is not a practical way to identify the new position of the tracked object. Therefore, minimum and maximum values should be defined for each component in the average touched pixel. Color radius for range checking in the HSV color space is used for each component. We use a radius of 25 for hue and 50 for both saturation and value. Instead of comparing the pixels in the processed frame with only one value for each component (average touched pixel), we compare with a range of values for each component. The result of this step is having lower bound and upper bound values for the three components of the average touched pixel.

When the second frame is ready to be processed to identify the new position for the tracked object, the following steps are applied on the frame:

1. The frame is downsampled twice by rejecting even rows and columns.

2. The result of step 1 is converted to HSV color space.
3. "In Range" [1] function is applied on the result of step 2. A function called "in Range" produces binary image where 1 means the pixel's component values lay in the range of the upper and lower bounds of the average touched pixel and 0 means the pixel's component values don't lay in that range.
4. The binary image is dilated.
5. The contours are found.

The contour that has the maximum area will be selected as the next position of the object. The aforementioned steps are applied for each frame. The whole tracking algorithm is summarized as flowgraph in Figure 8.

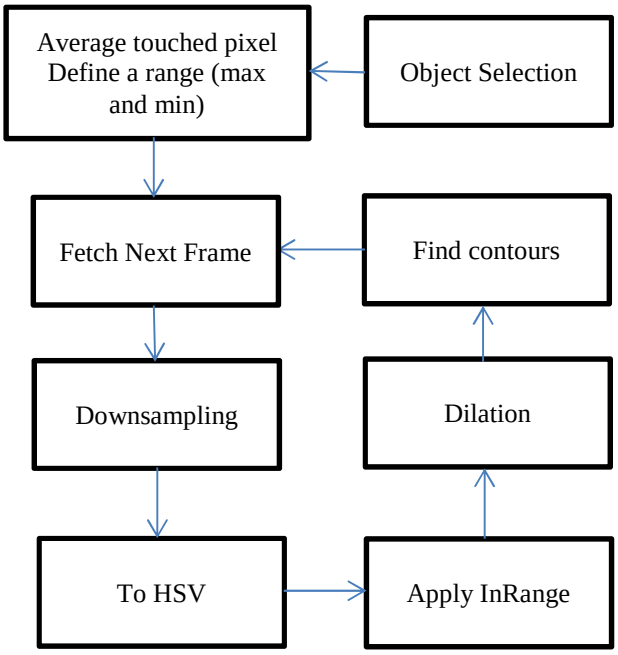


Figure 8: The flowgraph representation of color based tracking method.

B. Template Based Tracking Method

The previous method can only track a fully colored object. The method discussed in this section overcomes the previous method since it can track any object regardless its color and shape. The method starts from the first frame when the user fits the object that is wanted to be tracked in a template (rectangle) on the android device screen and decides to start tracking the object. The original template of the will be saved for future use in next frames. The next frames will go through three steps: object localization, object scaling handling and template adaptation.

1. Object Optimized Localization Using Powell's Gradient Ascent Method

The template based tracking method uses Powell's direct set method [2] for optimized localization of object in every frame since the brute-force search method, which uses Image Similarity Measure (ISM), is computationally complex and inefficient as well. To reduce the number of ISM operations, we use Powell's gradient ascent method for optimizing the object search. Many tracking systems [4] and medical image registration [3] use the same method. The steps of object localization are giving as follows [4]:

- **Step 1**: Select rectangular target region (Template, T) in the first frame and let (x_0, y_0) be its center and hx and hy being its width and height, respectively.
- **Step 2**: Initialize step-size, $\sigma = 3$ and fetch the next frame.
- **Step 3**: Compute Mean Absolute Difference (MAD) as given in Equation 1 between the template T and each of the five candidates (C) by shifting center of the rectangle to five position: (x_0, y_0) , $(x_0 \pm \sigma, y_0)$, $(x_0, y_0 \pm \sigma)$. Let C_{min} be the candidate that has the minimum MAD and let (x_1, y_1) represents its center.
- **Step 4**: If (x_1, y_1) is same as (x_0, y_0) , then

- If σ is greater than 0, reduce σ by one and go to Step 3.
- If σ is equal 0, it means the object is localized by the most likely target-region. The template may need to be processed through scaling and adaptation as described in the subsequent sections, then go back to Step 2 for next frame.

If (x_1, y_1) is not equal to (x_0, y_0), then set (x_0, y_0) equal to (x_1, y_1) and go back to Step 3.

$$MAD = \frac{\sum_{x,y}|T(x,y)-C_i(x,y)|}{hx \cdot hy}; i = 1,..,5 \qquad (1)$$

2. Object Scale Handling

Object Scaling is performed only if the Normalized Cross Correlation (NCC) between template, T, and selected candidate region, C_{min}, is above a threshold which was fixed to 0.8. The Normalized Cross Correlation (NCC) is given by Equation 2 where μT and μC_{min} are the mean intensity values of template, T and selected candidate region C_{min}, respectively. Scaling can be done as follows:

- **Step 1**: Scaling hx and hy of the selected candidate region by $\pm 5\%$. The aspect ratio and the center of the selected candidate region must be the same as before.
- **Step 2**: Resizing template, T, to match the rescaled candidate regions.
- **Step 3**: Compute MAD between resized templates and each corresponding rescaled candidate regions.
- **Step 4**: The one that minimizes MAD will be selected and also the template, T will be replaced with corresponding resize template.

$$NCC = \frac{\sum_{y=0}^{hy-1}\sum_{x=0}^{hx-1}[T(x,y)-\mu T][C_{min}(x,y)-\mu C_{min}]}{\sqrt{\sum_{y=0}^{hy-1}\sum_{x=0}^{hx-1}[T(x,y)-\mu T]}\sqrt{\sum_{y=0}^{hy-1}\sum_{x=0}^{hx-1}[C_{min}(x,y)-\mu C_{min}]}} \qquad (2)$$

3. Template Adaptation

As the case in object scaling, template adaptation is performed only if NCC is above a threshold which was fixed to 0.8. Template adaptation is an important task in tracking objects that change their appearance. So template adaptation will try to keep the template, T, up-to-date. Template adaptation can be done using Equation 3 which is a weighed sum of resized template, T, and selected candidate region, C_{min} where α for our simulation was fixed to 0.9.

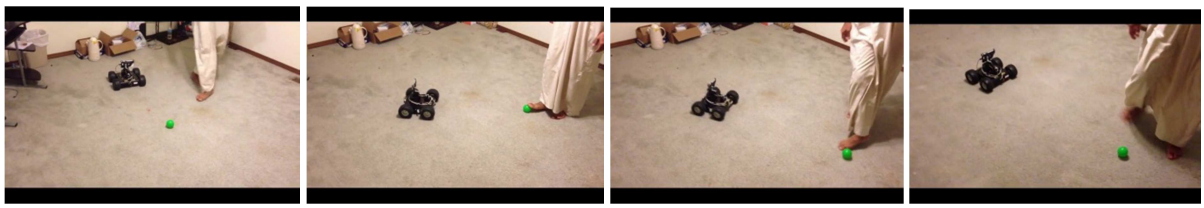
$$T(x,y) = \alpha T(x,y) + (1-\alpha)C_{min}(x,y) \qquad (3)$$

VI. EXPERIMENT

In this section, we present the experimental results of our robot by tracking different objects using the tracking methods discussed above. The source code of the libraries implemented in this project was uploaded to Google code and linked in [C1]. A video made from one of the experiment in this section was also uploaded to YouTube and linked in [C2].

A. Using Color Based Tracking Method

This method was implemented in OpenCV for Android and experimentation has been performed on the robot that uses a Galaxy Note (android device with 1.4 GHz dual-core processor) to record and process the frames. The robot can track any fully colored object using the color based tracking algorithm at an average frame rate of 25 frames per second, which is sufficient for real-time applications. Figure 9 shows a sequence of frames while the robot is tracking a green ball. Figure 10 shows a sequence of frames while the robot is tracking a blue ball.



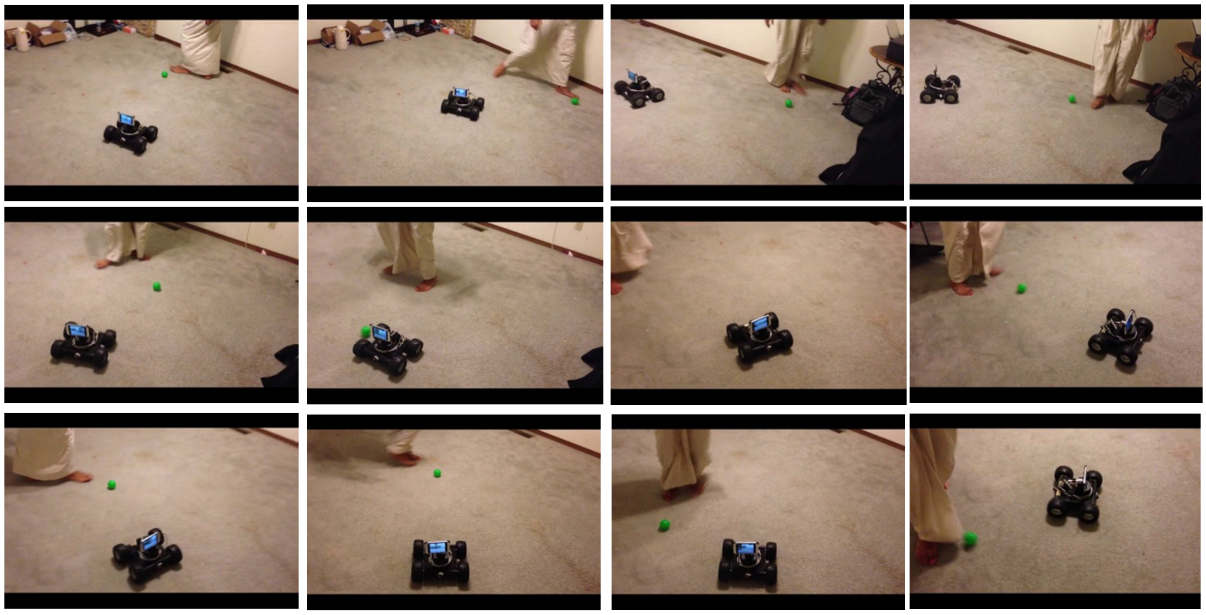


Figure 9: Sequence of frames of tracking a green ball

Figure 10: Sequence of frames of tracking a blue ball

B. Using Template Based Tracking Method

When we run the robot to track objects using template based tracking algorithm, the robot sometimes loses the tracked object. The reason is that the template based tracking algorithm demands more

powerful processor than the Galaxy Note's processor. The robot would not lose the tracked object only if the object moves slowly.

To test the template based tracking algorithm further, we run it on a PC with a 3.2 GHz Core i7 processor, a 9 GB RAM and a webcam for recording. The algorithm can track any rectangular target region specified by the user at an average frame rate of 80 frames per second, which is sufficient for real-time applications. The red rectangles in the following figures are the templates that need to be tracked. Figure 11 shows a sequence of frames where the tracked object is a toy car controlled by radio frequency remote control. Figure 12 shows a sequence of frames where the tracked object is a human eye. Figure 13 shows a sequence of frames where the tracked object is a toy rabbit.

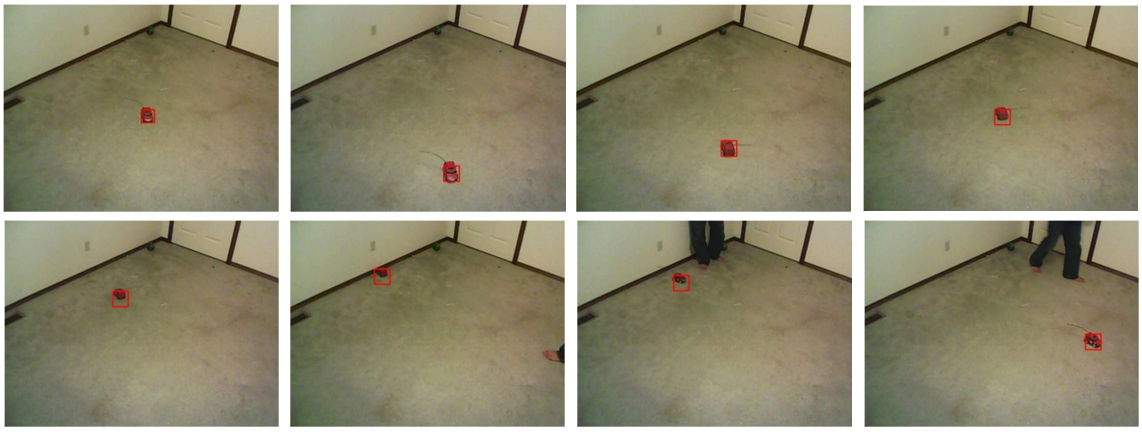


Figure 11: Sequence of frames of tracking a toy car

Figure 12: Sequence of frames of tracking a human eye

Figure 13: Sequence of frames of tracking a rabbit toy

VII. CONCLUSION

This paper describes a four wheeled robot built with an Arduino microcontroller controlled by an Android device for tracking and following moving objects. Android and Arduino libraries are implemented to allow an Android device to control the robot. The Android device is also used to process the images acquired through its camera. For image processing in the purpose of tracking moving objects, two different kernel based trackers are implemented as Android applications: color based tracker and template based tracker. The experimental results of the robot show robust tracking of a variety of objects undergoing significant appearance changes, with a low computational complexity.

REFERENCES

[1] OpenCV for Android, version 2.4.2. http://opencv.org/

[2] M. Powell, “An Efficient Method for Finding the Minimum of a Function of Several Variables without Calculating Derivatives”, In Computer Journal, vol. 7, pp. 155-162, 1964.

[3] X. Xu and R.D. Dony, “Differential Evolution with Powell’s Direction Set Method in Medical Image Registration”, In IEEE International Symposium on Biomedical Imaging: From Nano to Macro, Canada, vol. 1, pp. 732-735, 2004.

[4] S. Sarwar, M. F. Khan and N. Rao, “Real-time Object Tracking using Powell’s Direct Set Method for Object Localization and Kalman Filter for Occlusion Handling” , In Proc. of International Conference on Digital Image Computing: Techniques and Applications DICTA, Australia, 2012.

[5] J.F. Engelberger, “Health-care Robotics Goes Commercial: The Helpmate Experience”, in Robotics, vol. 11, pp. 517-523, 1993.

[6] L. Davis, V. Philomin and R. Duraiswami, “Tracking Humans from a Moving Platform” , in Proceedings of the International Conference on Pattern Recognition, vol. 4, p.4171, 2000, IEEE Computer Society.

[7] M. Bohmer. “Beginning Android ADK with Arduino”, 2012. Apress.

[8] R. Gonzalez and R. Woods. “Digital Image Processing”, 3rd, 2008. Prentice Hall.

[9] O. Javed and M.S. Yilmaz, “Object Tracking: A survey”, ACM Journal of Computing Surveys, vol. 38, 2006.

APPENDIX A (MAJOR COMPONENTS)

[1] Lynxmotion - A4WD1 Chassis

[2] Lynxmotion - Off Road Robot Tires

[3] Arduino - Arduino ADK R3

[4] Pololu - 50:1 Metal Gearmotor 37Dx54L mm with 64 CPR Encoder

[5] Sabertooth - Dual 12A Motor Driver

[6] MaxBotix - MB1000 LV‑MaxSonar EZ0

[7] Lynxmotion - Base Rotate Kit

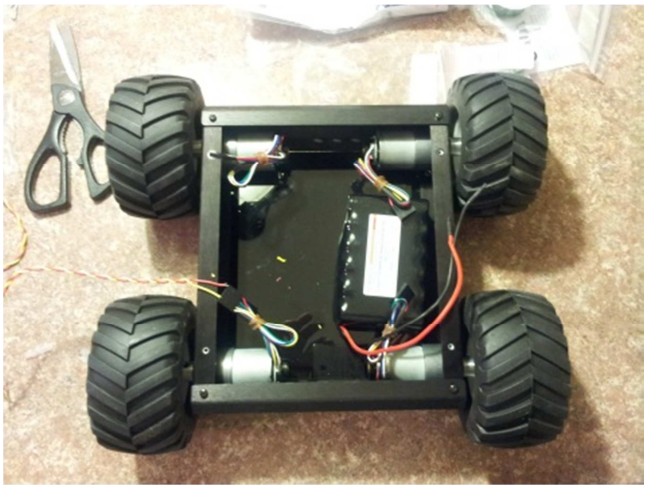
[8] Lynxmotion - Pan and Tilt Kit

[9] Hitec Robotics - S-422 Servo Motor

[10] Tenergy Coporation - Li-Ion 18650 11.1V 10400mAh Rechargeable Battery Pack

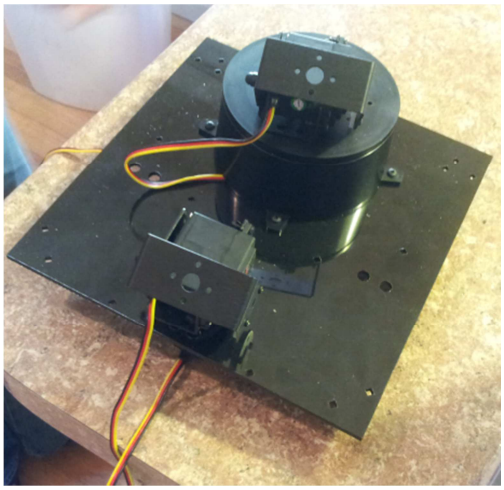
APPENDIX B (PICTURES)

[1]



[2]

[3]



APPENDIX C (SOURCE CODE AND EXPERIMENT VIDEO)

[1] The Source Code of our project

http://code.google.com/p/android-arduino-object-tracking-robot/

[2] One of the Videos of the Robot Experiments

http://youtu.be/mw__gOjdyYI