BRADLEY
University

Distributed Vision-Based Target Tracking Control Using Multiple Mobile Robots

Anthony Le and Ryan Clue

Supervisors:
Dr. Jing Wang & Dr. In Soo Ahn

This thesis is presented as part of the requirements for the conferral of the degree:

Bachelor of Sciences

Bradley University
Department of Electrical and Computer Engineering

May 5, 2017

Abstract

In this project, a distributed vision-based control system is designed for multiple mobile robots to address the target tracking problem while maintaining the specified formation among robots. The system mainly consists of two modules. One is for target identification and the other is for target tracking control. In the target identification module, the robot is controlled to pivot around its center and perform a survey of the environment using its on-board vision sensor. In the target tracking control module, a leader-follower control strategy is adopted to solve the target tracking and formation control problem of multiple robots. The proposed distributed vision-based control is experimentally tested using three QBot2s from Quanser, Inc.

Acknowledgments

Mr. Mattus: For setting up our development environment and figuring out how to create an ad-hoc network in Windows 10
Dr. Lu: For his assistance in creating this presentation as well as helping us over the occasional stumbling block in Simulink

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Distributed control of multiple unmanned autonomous robots (UARs) has been the subject of intensive research in recent years due to the potential applications in search and rescue missions, surveillance and reconnaissance, environmental sensing and monitoring, intelligent transportation and threat isolation/evasion, and obstacle evasion [5]. There are many practical commercial, civil and military applications that may benefit from advancements in UAR control strategies and algorithms from this area of research.

1.2 Related work

Methods for successfully coordinating multiple robots to autonomously encircle a target point have been put forth by several research papers. These distributed controls assume communication the robots to communicate their kinematic state variables between one another, and consequently researchers have designed their algorithms in ways that minimize the number of communication channels required between robots. However, there has not been nearly as much research done that attempt to implement coordinated encirclement that rely entirely on each individual robots vision sensors.

A review of prior work at Bradley and research literature was conducted by our team during the conception phase of the project. This included a review of e prior Class of 2016 Senior Project, Cooperative Control of Heterogeneous Mobile Robots Network [5]. The 2016 research and capstone project experience using the QBot2 robots was leveraged in our project. The 2016 project tested cooperative control using the QBot2 platform including the Qbot2 Camera and the Kinect RGBD camera for localization. Wifi communication was used to exchange position information among robots. Consensus and formation stabilization problems were studied.

Our project also researched and references another work, highly relevant to our 2017 project objectives. Decentralized multi-robot encirclement of a 3D target with guaranteed collision avoidance for object tracking and encirclement by Franchi, Stegagno, and Oriolo [3]. Their work on 3-D (and 2-D) and distributed collaborative control is extensive and leverages research going back to 2001. Their findings delivers a control framework for achieving encirclement of a target moving in a 3D space using multiple robots, image-only sensing, and requiring local communication between exactly two other members of the encircling group of robots.

An additional consideration, for a vision-only sensing strategy was image processing techniques and algorithms. We relied on Peter Corkes book "Robotics, Vision and Control" [10] as well as his Matlab toolbox for much of our image processing. The academic tutorials packaged with the Qbot2 were also valuable resources. It is worth mentioning that systems in the literature utilizing visual-only sensing generally assumed 360-degree robot sensing (e.g. fisheye camera). Wang et al.'s research [3] utilized a 240-degee field-of-view sensor and a laser range finder for identification and locating targets to support inputs to their control strategies. Because the Qbot2 cameras have less than a 90-degree field-of-view, our design would have to incorporate additional logic to locate targets outside the robots visual range.

Chapter 2

Problem Formulation

2.1 Kinematic Model of Mobile Robots

Depending on whether the robot is following a target or leader robots, two separate control modules are used. Both modules use a Cartesian coordinate system and follow a target point, which is obtained by transforming the current or previous result of the image processing module with respect to the robots current position.

2.2 Vision-based control

The purpose of the vision-based control is to provide a target point to the kinematic control, which it accomplishes by processing the Qbot2s color images and depth images. This control should also provide additional information as to whether the target point it provides should be considered valid or not.

2.3 Project Objectives, Goals, and Approach

The goal of our project is to design distributed vision-based control algorithms for mobile robots and to implement and validate the proposed algorithms. The primary project tasks (research, concept definition, design, and demonstration) tasks included:

- Design a target identification (detect and locate) module based on RGB image features obtained from a vision sensor
- Design a target tracking algorithm based on robot model linearization
- Design a leader-follower formation control algorithm based on depth and image features provided from the target identification module

- Design a state machine to coordinate target identification module and target control module and communications between Robots
- Integrate and validate the proposed distributed controls through experimentation in a controlled lab environment

2.4 Project Assumptions

The lab environment, limitations of the research and practical limitations (and availability) of fully operational equipment are also considered.

- The lab environment will be a well-lit, indoor area
- The target will remain on the same level plane as the robots (2D motion tracking).
- The target will not move faster than the Qbot2s maximum velocity (0.7 m/s).
- The target will be a solid colored, basketball-sized sphere.
- The environment will be reasonably free of similarly colored objects.
- As stated above the multiple robots must use same control strategy.

Chapter 3

Design of Distributed Vision-based Control

3.1 Design approach

The basic system is comprised of the QBot2 Mobile Robot and QUARK Control Software on a host computer. The full system is comprised of multiple QBot2s collaboratively tracking a single target with the Quark Control Software Computer. Note: An assumption is that the Quark Control Software on the host computer could be ported entirely to the robots, given sufficient processing and communications, to attain higher autonomy and actual decentralization of the system. The target in the experiment is a round ball that is detected, located, and tracked by the multiple QBot2s. Subsystems for the basic system, a single QBot2, include: the Qbot2 mechanical/motors/control subsystem, the sensing/image processing subsystem, and the communications/processing subsystems (on the Qbot2 and the Host computer).

3.2 QBot2 Mobile Robot and QUARC Platform

The QBot2 utilizes QUARCs HIL software suite to allow users to design their program using Matlab and Simulink. The Simulink model and Matlab code are translated to C++ code by the QUARC software and compiled automatically on the Qbot2s computer.

3.2.1 Hardware

The QBot2 comes equipped with a differential drive robot base which has a maximum speed of 0.7m/s. A v.1 Kinect-for-Xbox mounted on the robot base provides both color and distance vision information. The RGB image that has a default resolution of 640 × 480 pixels and a 57 × 43 field of view. The depth image has a resolution of 640 × 480 pixels, approximately 52.5 × 41 field of view, and measures distances from 0.5m to 6.0m. It is worth noting that the accuracy of the depth map depends on the detail of the grid projected by the IR emitter, the actual accuracy of the depth image is significantly lower than a resolution of 640x480 implies. QBots use IEEE 802.11 b/g/n protocol for communication between the computer and QBot2. TCP/IP protocol is utilized, with each QBot2 using a statically assigned IP address.

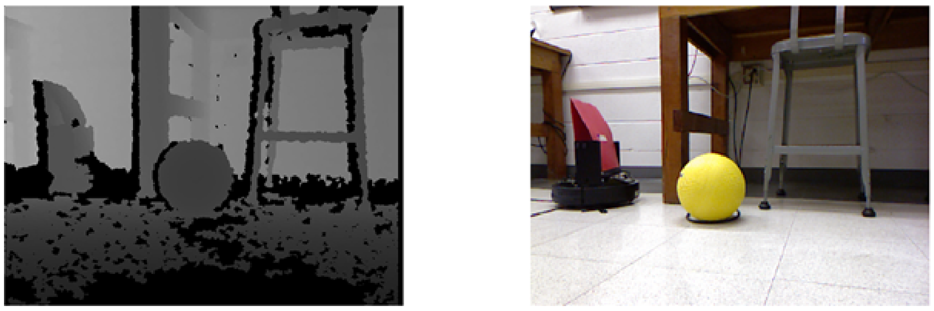


Figure 3.1: Side-By-Side of Color and Depth Images

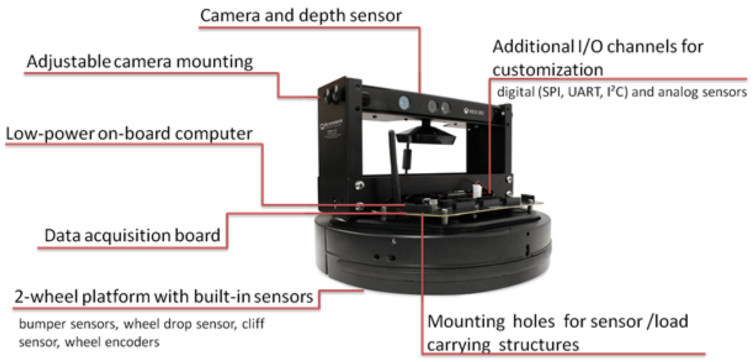

Camera and depth sensor
Adjustable camera mounting
Additional I/O channels for customization
digital (SPI, UART, I²C) and analog sensors
Low-power on-board computer
Data acquisition board
2-wheel platform with built-in sensors
bumper sensors, wheel drop sensor, cliff sensor, wheel encoders
Mounting holes for sensor /load carrying structures

Figure 3.2: QBot2 Mobile Robot

3.2.2 Software

To connect to the QBOT2 A host-target real-time control system [2] is utilized. As shown in Figure 3-2, the target QBOT2 computer is connected wirelessly with the host computer on which the SIMULINK model is running. The control algorithms are developed in MATLAB/SIMULINK with QUARC on the host computer. The control models are cross-compiled and downloaded to the target computer in real time.

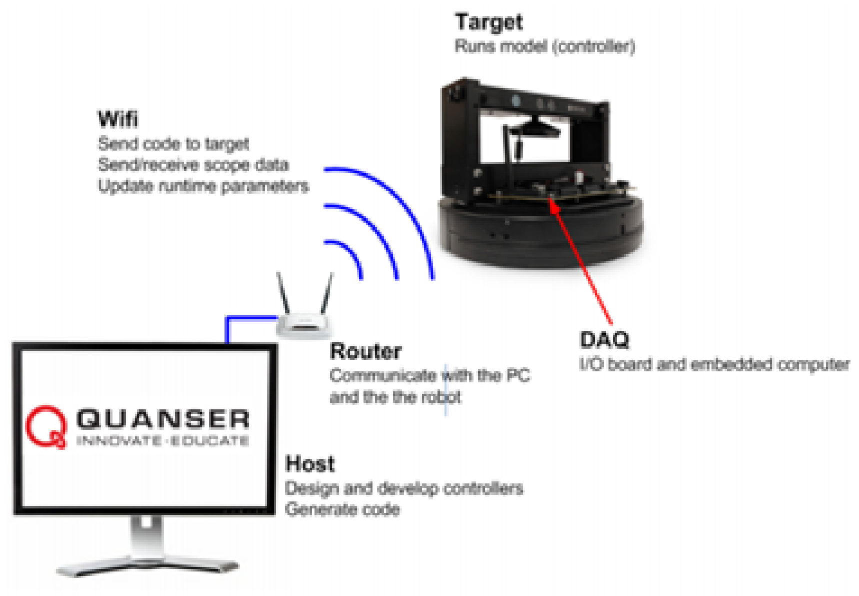


Figure 3.3: Host Computer / Wi-Fi Router to Qbot Communications Connectivity

3.3 Target Identification

3.3.1 Interface and inputs to Target Control

The result of the vision control system is a target location given in 2D Cartesian coordinates. The origin of this coordinate plane is at the Qbot2s camera position, the positive X axis extends in same the direction the Kinect is facing and the positive Y axis extends to the left (or port side) of the Kinect. The motion control module will then transform this point according to its current kinematic state. This is target point is the basic interface between the target identification module and the motion control.

Based on the 2016 Senior Project, and research of alternatives, we chose to use the color image for object/target recognition and the depth image for determining distance to the target. Our choice was driven primarily by practical QBot2 processing limitations and timing restrictions prohibiting potentially more robust methods of object detection.

3.3.2 Acquire Y Component of Target Location from Color Imagel

We perform all object recognition on the color image. Consequently, color image processing is the primary bottleneck in system performance. We initially selected color thresholding with blob detection because it is the easiest image processing method to implement. Although we did pursue alternative methods of image processing, these implementations were unable to meet our performance requirements due to limitations of the Qbot2 hardware. Therefore, color thresholding with blob detection was selected because it had a significantly lower performance requirement while also producing adequate results.

The first stage of image processing, color thresholding, highlights all elements of the image that correspond to our targets color. During this stage, pixels with RGB values that fall within a pre-determined range are replaced by 1 in the binary output image, while all other values are replaced by 0.

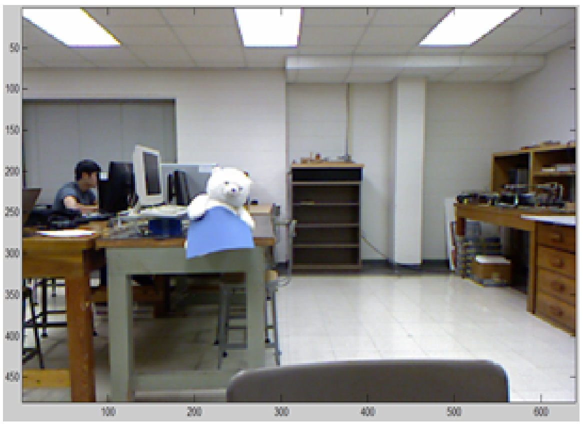


Figure 3.4: Before Processing - Color Thresholding and Blob Detection

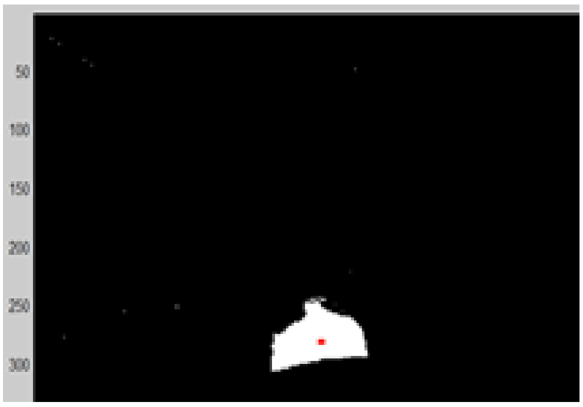


Figure 3.5: After Processing - Color Thresholding and Blob Detection

Blob detection is applied to the binary output image from color thresholding. The goal is to identify the largest grouping of neighboring, non-zero values. Then, for each group (or blob) of neighboring pixels, we calculate its centroid by finding the mean index of all pixels contained in that blob. This means that the accuracy of the blob corresponding to our target object is unaffected by noise from similar colors appearing elsewhere in the image. It should be noted that the accuracy of this method is therefore entirely dependent on the color thresholding result.

The blob detection outputs a list of centroids and corresponding blob sizes. If the blob size of the largest blob is greater than some pre-determined threshold value, our system will use that centroid for the targets coordinates on the RGB image plane.

3.3.3 Acquire X Component of Target Location from Depth Image

The targets location on the depth map can be estimated by translating points from the color image plane to the depth image plane. We calculated this using the ratio between the two images field-of-view and the estimated angle from the RGB image.

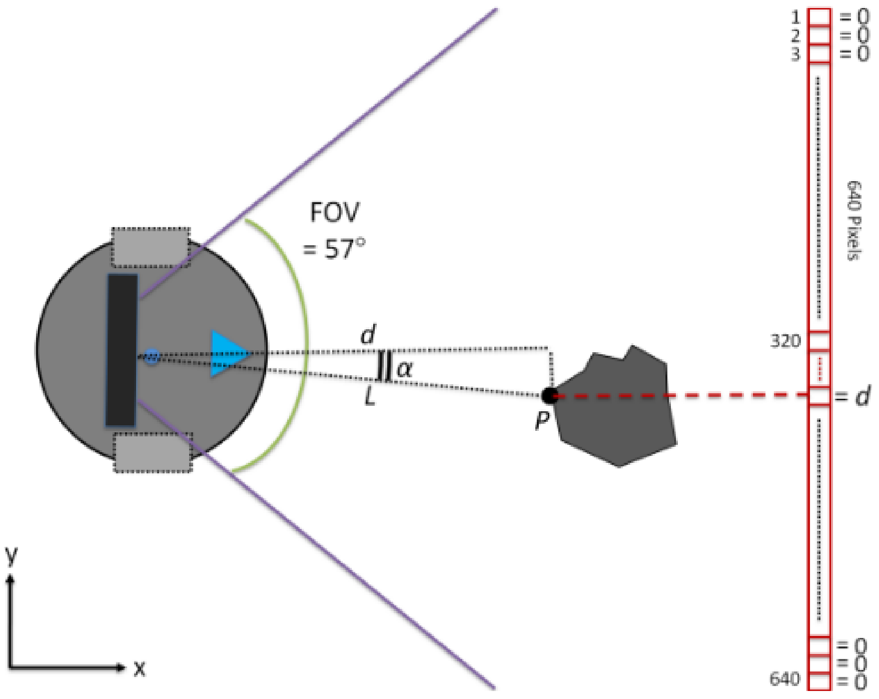


Figure 3.6: Target Localization - Finding Distance

$$ {}_T = (L_{color}/2 - X_{color})((FOV_{color} - x)/L_{color}) \tag{3.1}$$

$$ XY_{depth} = (FOV_{color}/L_{color})(L_{depth}/FOV_{depth})XY_{color} + offset \tag{3.2}$$

$$ X_T = Image_{depth}(X_{depth}, Y_{depth}) \tag{3.3}$$

$$ Y_T = X_T tan({}_T) \tag{3.4}$$

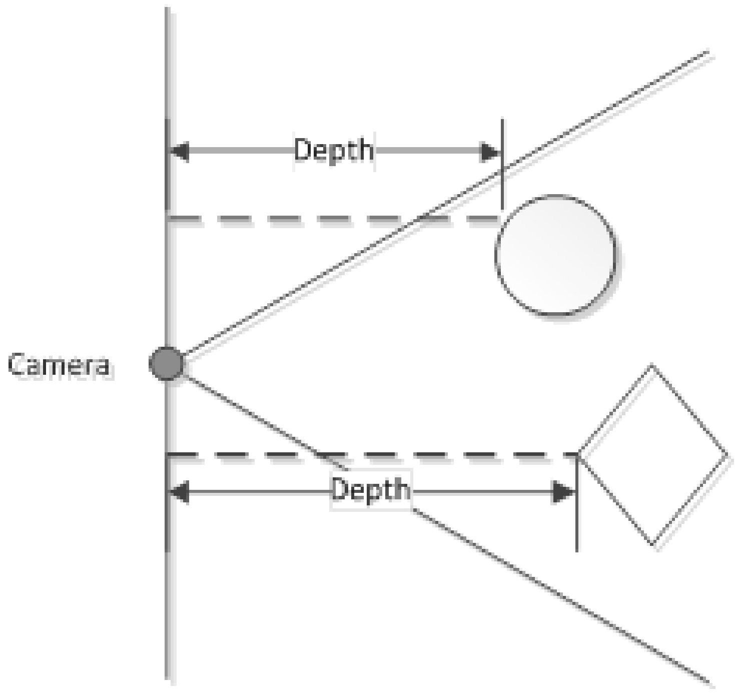


Figure 3.7: Target Localization - Determining x,y coordinates

3.4 Motion Control

3.4.1 Encirclement

The target control module relies on received target coordinates from the image processing module and the robots x,y, and theta. These are inputs which allow for formulation of encirclement. From the robots x,y, and theta values, we transform these values to the appropriate kinematic model for the QBot2. From this kinematic model, we transform the cartesian local coordinate system to a cylindrical coordinate system. The length between the two wheels of the QBot is 0.0235 meters. We use the stardard kinematic model,

$$\dot{x} = V_c * cos(\theta) \tag{3.5}$$

$$\dot{y} = V_c * sin(\theta) \tag{3.6}$$

$$\dot{\theta} = \omega_c \tag{3.7}$$

to find the forward point of the robot,

$$P_x = x + 0.0235 * cos(\theta) \tag{3.8}$$

$$P_y = y + 0.0235 * sin(\theta) \tag{3.9}$$

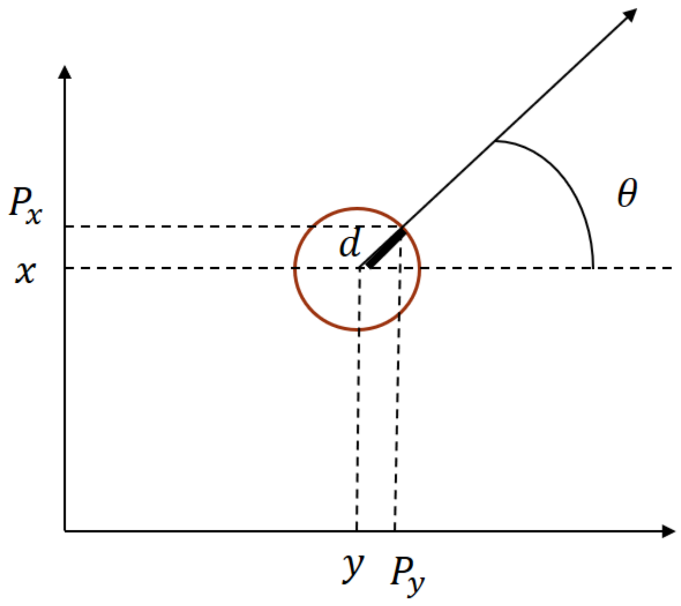


Figure 3.8: P_x and P_y Conversion

Following, we want to define p_i,

$$p_i = (\rho_i, \phi_i) \tag{3.10}$$

and convert these values by,

$$\rho(P) = \sqrt{(P_{xd}^2 + P_{yd}^2)} \tag{3.11}$$

$$\phi(p) = tan^{-1}(P_{yd}, P_{xd}) \tag{3.12}$$

where P is the pose of the model.
$\dot{\rho}$ and $\dot{\phi}$ are the forward velocity and angular velocity computed by: —-

$$\dot{\rho} = k_r * (\rho_s - \rho_d) \tag{3.13}$$

$$\dot{\phi} = \omega_d \tag{3.14}$$

where k_r is the porportional gain and ρ_s is the size of the encirclement radius. ρ_d is the approximate distance from the robot to the target given from the inverse kinematics and image processing modules.

We then use the Jacobian matrix of the difference between the target and robot where P_x and P_y is the X and Y coordinates of the robot and P_{xt} and P_{yt} is the X and Y coordinates of the target in relation to the robots coordinate frame. The difference of the robots X and Y locations can be reduced to P_{xd} and P_{yd}. and the following Jacobian matrix can be computed

$$J_i = \begin{bmatrix} P_{xd}/\sqrt{(P_{xd}^2 + P_{yd}^2)} & P_{yd}/\sqrt{(P_{xd}^2 + P_{yd}^2)} \\ -P_{yd}/(P_{xd}^2 + P_{yd}^2) & P_{xd}/(P_{xd}^2 + P_{yd}^2) \end{bmatrix} \tag{3.15}$$

From the Jacobian, the following input velocities Ux and Uy can be computed

$$\begin{bmatrix} U_x \\ U_y \end{bmatrix} = J_i^{-1} * \begin{bmatrix} \dot{\rho} \\ \dot{\phi} \end{bmatrix} \tag{3.16}$$

For the velocity inputs, U_x and U_y , they are transformed using the inverse kinematic equations

$$p^{-1} = \begin{bmatrix} cos(\theta) & -0.0235 * sin(\theta) \\ sin(\theta) & 0.0235 * cos(\theta) \end{bmatrix}^{-1} \tag{3.17}$$

to find V_c and ω_c,

$$\begin{bmatrix} V_c \\ \omega_c \end{bmatrix} = p^{-1} * \begin{bmatrix} U_x \\ U_y \end{bmatrix} \tag{3.18}$$

Once V_c and W_c are found, using

$$V_r = V_c + 1/2 * .0235 * 2 * \omega_c \tag{3.19}$$

$$V_L = V_c - 1/2 * .0235 * 2 * \omega_c \tag{3.20}$$

we can find velocities of the right and left wheels, V_r and V_L.

Right and left wheel velocities are then sent to the HIL Write block which sends data to the 2000 and 2001 ports of the QBot2 for the left and right wheels, respectively. The QBot2 stores wheel encoder data to allow calculation of actual right and left wheel velocities. This data is taken from the HIL Read block. For each wheel encoder ticks, we can determine the velocity of each wheel. The robots x,y, and θ values are then updated and sent back into the control module using V_c and θ where,

$$V_c = 0.5 * (V_L + V_r) \tag{3.21}$$

$$\theta = 1/0.235 * (V_r - V_L) \tag{3.22}$$

From V_c and ω_c, we can determine the robots X, Y, and θ values by calculating the integrals for

$$\dot{x} = cos(\theta) * V_c \tag{3.23}$$

$$\dot{y} = sin(\theta) * V_c \tag{3.24}$$

$$\dot{\theta} = \omega \tag{3.25}$$

3.4.2 Leader-Follower

Leader-follower control module is used for distributed multi-robot coordination. The leader-follower control module does not take into account inverse kinematics. And, the image processing is used to localize the leader robot coordinates (x_r , y_r) in the vision sensor range. Given the radial distance of the target, V_c and ω_c are calculated. A follow distance of 0.5 meters and gains Kw = 0.5 and Kv = 0.2 were defined as per limitations for the QBot. To determine V_c and ω_c,

$$V_c = K_v * \sqrt{x_t^2 + y_t^2} - 0.5 \tag{3.26}$$

$$\omega_c = -K_w * tan^{-1}(x_t, y_t) \tag{3.27}$$

where x_t and y_t are the positions of the target in relation to the robots local coordinate frame. One should note the limits for angular and forward velocity are set to .2 m/s.

3.5 Event-based System Control

3.5.1 Implementation

The Simulink model is configured using the HIL block from the QUARC toolbox. This enables users to more easily run their simulations in a virtual environment as well as on actual hardware. The project must also be configured to communicate with a specific IP address, which can be done through the QUARC add-on options menu in Simulink. It is also important to set the sample time configure the Solver pane of the Simulink model configuration. It is important to select the ode1 and fixed-step options. Additionally, when performing operations on images or large arrays of data, it is recommended that users select a sample frequency no greater than 15Hz. Our experiments used a 10Hz sample frequency.

3.5.2 Tasks Coordination

Our system would need to switch between two modes of operation depending on whether or not the target was in view. Stateflow was used to control the varying states in our model. Models for both target encirclement and leader-follower used similar simulink and stateflow models with differences only in controls and how often a search was made. In both models, search mode is done by toggling between image processing and rotating. Once the desired accuracy is obtained from blob detection, the state switched to the control module. The target encirclement model only searches for target every set time-interaction whereas leader-follower will search for a target every other time-step. To view the entire Stateflow and its subsystem, see figure B.1 to B.3 on page 42 to 44.

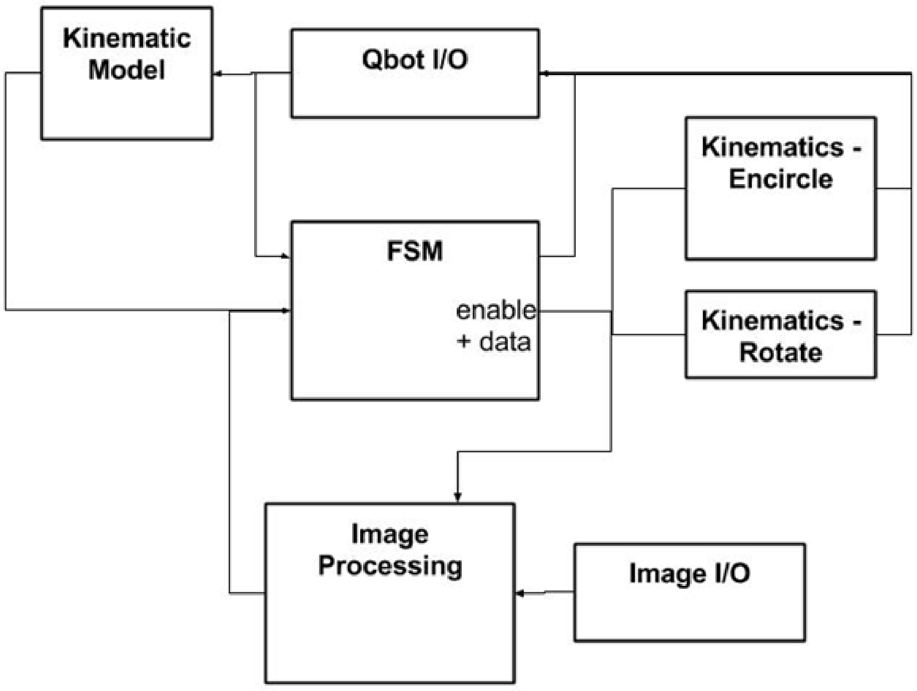


Figure 3.9: Abstracted View of Simulink Diagram

3.5.3 Target Encirclement - Finite State Machine

In the stateflow for target encirclement control, various enable switches allow triggering of different modules. The system starts off with all enable switches off. While entering target acquisition mode, image processing module is enabled and determines if an image centroid can be made. If no centroid is found, the QBot2 will rotate 15 degrees on its axis and wait for the cameras to update. If a centroid can be made, it checks whether the pixel size of the image is greater than a certain value. In our case, a yellow ball or box is used. If 30 pixels are connected, the robot will continue into the encirclement module. Data from the image processing, such as the robots x, y, and theta, is also sent to the encirclement module. From this module, the target location is stored and encirclement is based on this value. After 120 seconds, the encirclement ceases and target acquisition mode begins again.

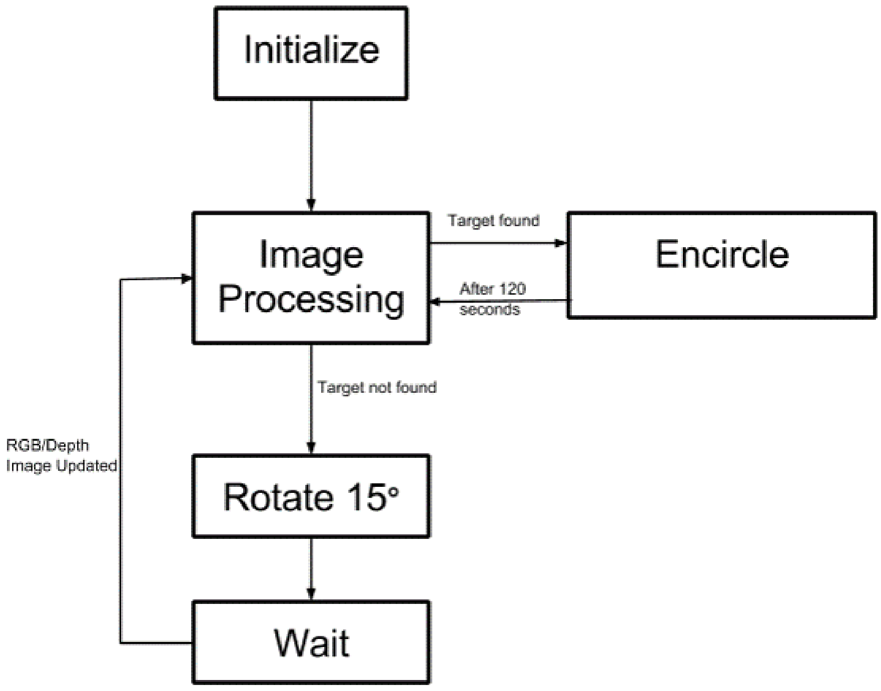


Figure 3.10: Event-based System Control: Encirclement Control

3.5.4 Leader-Follower - Finite State Machine

Leader-follower control is similar to target encirclement control. However, instead of returning to Target Acquisition mode, the target will continue to follow any centroid even if the pixel count is less than 30. The process is done by continually switching from image processing and the leader-follower control module. If no centroid is found, the target will return into Target Acquisition mode.

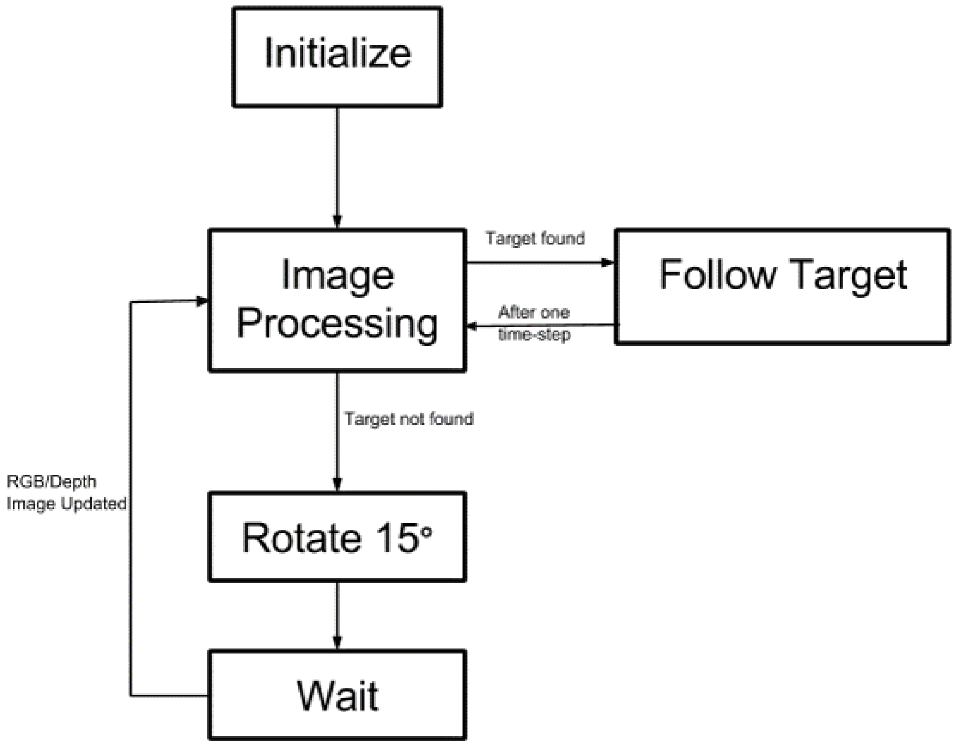


Figure 3.11: Event-based System Control:Ledaer-Follower Control

3.6 Experiments

A curriculum for the QBot was provided to allow interfacing and familiarity of the QBot2 and its software package, QUARC. The curriculum covers topics ranging from QBot communication, integration, kinematics and vision guided control. Advanced topics such as path planning, mapping, and localization are also covered. Each section provides an in-depth tutorial for the understanding of the QBots capabilities. Tutorials for blob detection, line-following, and kinematics were most referenced in our project. Various simulations for the control module were implemented in Matlab and Simulink. These demonstrations assumed constant communication between robots. For encirclement control, single robot to a multiple robot models were simulated to show proof of concept. Refer to the appendix for more information. From simulation, the first series of experiments were conducted. Using a predefined target point, encirclement of a single robot was tested. Our next iteration, using blob detection, allowed us to capture and encircle the target point. The next iteration consisted of determining which method of control was needed to allow coordination between robots. Stateflow was used to control different modules efficiently. Following, a leader-follower model was implemented for successive robots.

3.7 Results

Experimental results can be seen in figures below. Target encirclement was achieved by the target robot. Using the appropriate thresholds for the color yellow, the leader robot was able to identify the target. Also, the follower robot was able to correctly identify and follow the leader robot.

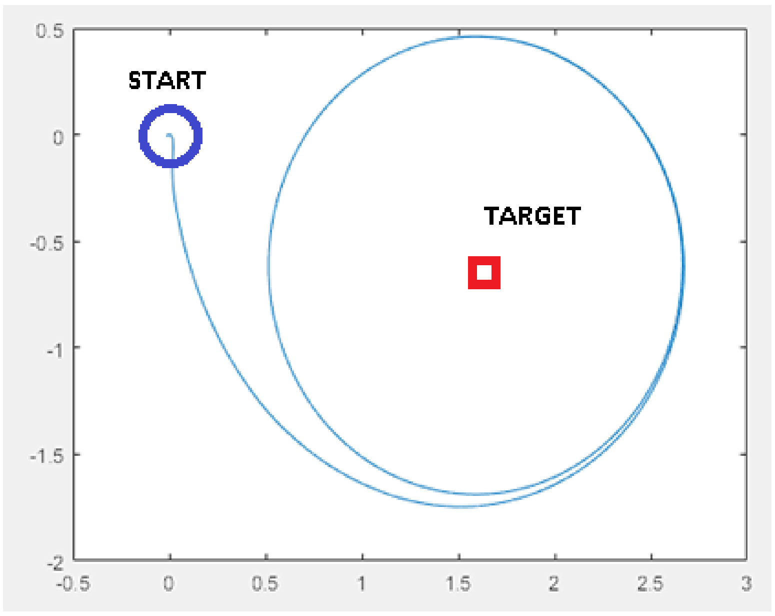


Figure 3.12: Phase Plot of Encirclement Robot

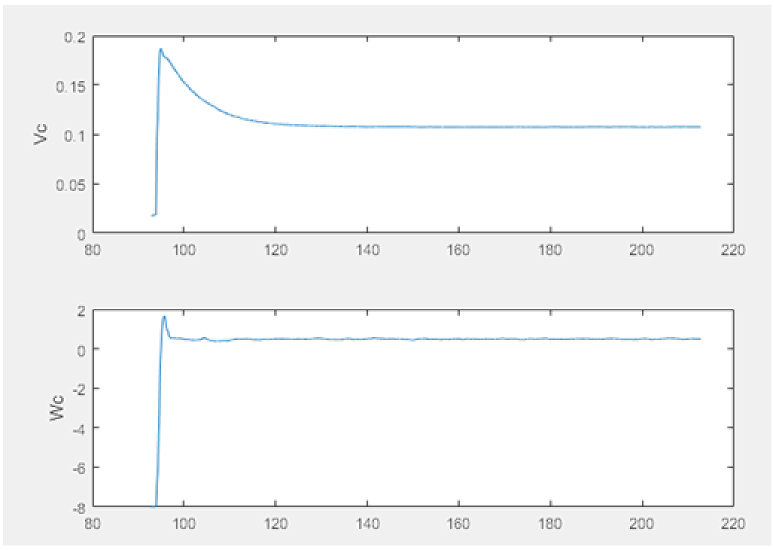


Figure 3.13: Forward and Angular Velocities of Encirclement Robot

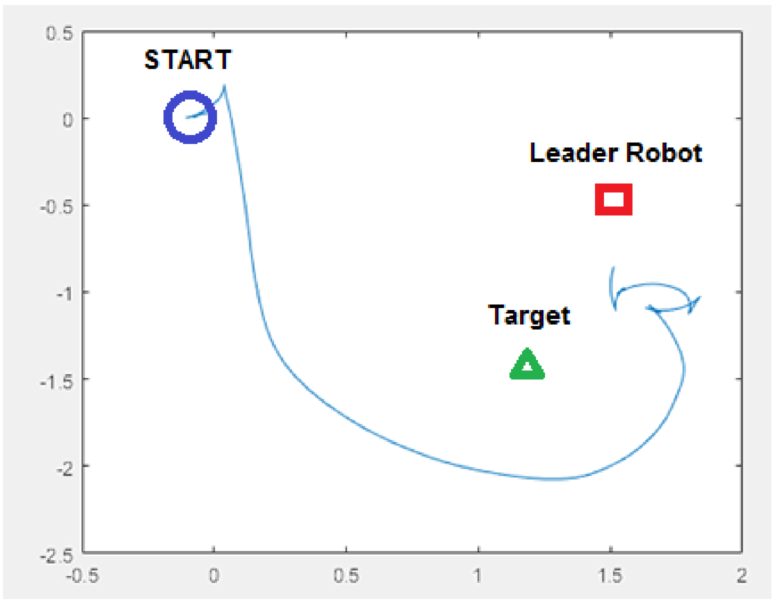


Figure 3.14: Phase Plot of Leader-Follower Robot

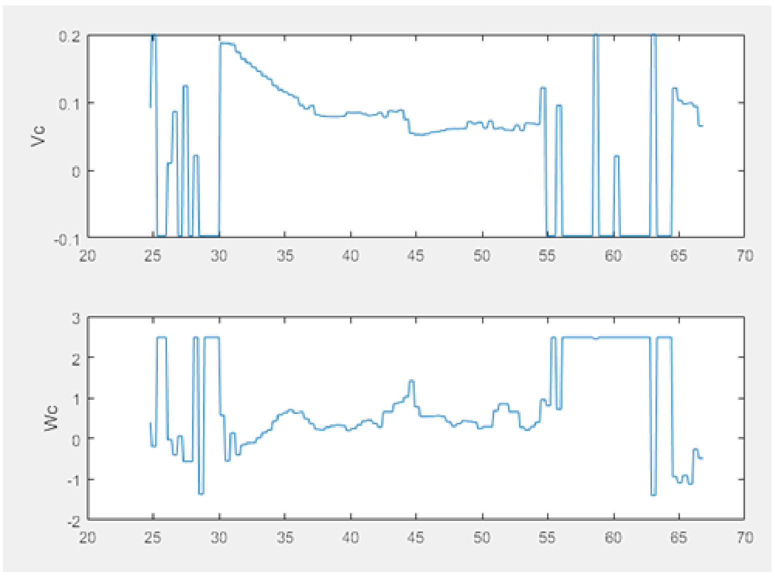


Figure 3.15: Forward and Angular Velocities of Leader-Follower Robot

Video Results:
Encirclement control can be seen in the following:
https://www.youtube.com/watch?v=VHpTNNhiLG4
Leader-follower control can be seen in the following:
https://www.youtube.com/watch?v=bO5DHXQCITw

Chapter 4

Project Management

4.1 Division of Labor

Ryan Clue implemented our image processing methods, created the initial Stateflow model, and provided the Simulink design for integrating the image processing modules with the kinematic modules.

Anthony Le implemented our kinematic control systems, created our Matlab data logging module, and extended our Stateflow model for the Leader-Follower program.

Anthony and Ryan worked jointly to debug and troubleshoot.

4.2 Project Schedule

For the first semester, we familiarized ourselves with interfacing the QBot2. In this semester, we debugged the Qbot and QUARC to interface on Windows 10 which is not readily supported. Following, we ran through tutorials that helped us understand the Simulink dynamics that included the kinematic controls and image processing of the QBot. Some tutorials worth noting are the Inverse Kinematic Tutorial, Image Thresholding Tutorial, and the Line-Following Tutorial. At the end of the first semester, Anthony focused primarily on the controls while Ryan focused on image processing. By the start of the second semester, creation of a simulation of encirclement and image detection was finished. During the second semester, implementation of encirclement and image detection on the QBot followed. At first, only separate modules were made and tested. During the middle of the semester, we were able to combine the modules using Stateflow.

Chapter 5

Discussion

5.1 Impact of the Design

Our design allows limited communication and sensing among all robots. Assuming a leader and its peripheral robots, we can obtain encirclement with each consecutive robot containing a smaller and smaller encirclement radii. This two part modular design allows for classifying the leader and followers and allows for increased modularity and customization if needed. The design was done by combining image processing and control task modules using Stateflow switched control strategies. With Stateflow, replacing a certain image processing or control module is as simple as replacing or rewriting a Simulink block. Creating simulations for certain environments can also be implemented using Stateflow. The overall design can be implemented in search and rescue missions, environmental surveying, and much more.

5.2 Future Work

There are two areas that can be significantly improved or developed.

The first improvement would be to implement collision avoidance. This would involve additional image processing as well as control logic. The kinematic control can be improved to accommodate obstacle avoidance by scaling the encirclement radius as the Qbot2 approaches the obstacle in its path (ref encirclement paper). As for identifying the obstacle, we believe that this can be accomplished by background subtraction - identifying expected distance from the floor and identifying all image elements that deviate from that expected floor distance.

The method of object recognition could also be improved. One possibility is using edge detection and performing a Hough Transform for the expected target object shape.

Further work could involve working with more robots to test the scalability of the algorithm.

Chapter 6

Conclusion

Our project have demonstrated distributed vision-based target tracking control using mobile robots. We have achieved the project goals mentioned in Section 2.3 that is to design a target identification (detect and locate) module based on RGB image features obtained from a vision sensor, to design a target tracking algorithm based on robot model linearization, to design a leader-follower formation control algorithm based on depth and image features provided from the target identification module, to design a state machine to coordinate target identification module and target control module and communications between robots, and to integrate and validate the proposed distributed controls through experimentation in a controlled lab environment. All the while, there were still problems with the upcoming design. The QBot2 does not contain enough processing power for the image processing that we desired therefore, blob detection was used. A better target recognition algorithm, such as hough transforms, allows a more robust target detection method. Failures, such as the target being misclassified or kinematic issues, were also noted and fixed during implementation. The Qbot occasionally would reset and move backwards once a target object was found. We believe this may be an issue with misclassification or a faulty kinematic model. Another problem with the QBot concerned the updating of RGB and IR images. The middleware for the Simulink design block that increased a tick for every image update occasionally would not change. This prevented us from keeping track each time image processing was finished and delayed search mode. Overall, this project demonstrated that minimized local communication and formation between robots is feasible. Future work, such as obstacle avoidance and improved image processing, can still be researched.

Bibliography

[1] Cap 02.pdf. [Online]. Available: http://mayerle.deps.prof.ufsc.br/private/eps6405/Cap 2002.pdf. [Accessed: 14-Nov-2016]

[2] Cooperative Control of Heterogenous Mobile Robots Network. [Online]. Available: http://ee.bradley.edu/projects/proj2016/mrn/. [Accessed: 14-Dec-2016].

[3] Franchi, Stegagno, and Oriolo. Decentralized multi-robot encirclement of a 3D target with guaranteed collision avoidance. 2016

[4] Freda and Oriolo. Vision-based interception of a moving target with a nonholonomic mobile robot. 2007

[5] G. Bock, R. Hendrickson, J. Lamkin, B. Dhall, J. Wang, and I. S. Ahn, Experiments of distributed control for multiple mobile robots with limited sensing/communication capacity, in 2016 IEEE International Conference on Electro Information Technology (EIT), 2016, pp. 05590564.

[6] G. Oriolo. Wmr control via dynamic feedback linearization: design, implementation and experimental validation. *IEEE Trans. on Control Systems Technology*, 10:835852, 2002.

[7] Kinect for Windows Sensor Components and Specifications. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131033.aspx. [Accessed: 14-Dec-2016].

[8] S. Miah, Lab 2: Trajectory Tracking Using Differential Drive Mobile Robots. 2016 .

[9] MRNProposalDocument.pdf. [Online]. Available: http://ee.bradley.edu/projects/proj2016/mrn/MRNProposalDocument.pdf. [Accessed: 14-Nov-2016]

[10] P. Corke, Robotics, Vision and Control - Fundamental Algorithms in MATLAB . Sep 2011

[11] Quanser - QBot 2 for QUARC. [Online]. Available: http://www.quanser.com/products/qbot2. [Accessed: 14-Nov-2016].

Appendix A

Appendix 1. Code

```
function [r, g, b, bin]  = clr_thr(red, ...
    green, blue, red_th, green_th, blue_th)
% This block supports the Embedded MATLAB subset.
% See the help menu for details.

red_vec = thresh(red, red_th, 1);

green_vec = thresh(green, green_th, 1);

blue_vec = thresh(blue, blue_th, 1);

tmp = double(and(red_vec, green_vec));
tmp_blob = uint8(and(tmp, blue_vec));
bin = uint8(255*tmp_blob);

% background = uint8(255*double(not(tmp_blob)));
% r = uint8(red.*tmp_blob + background);
% g = uint8(green.*tmp_blob + background);
% b = uint8(blue.*tmp_blob + background);

r = uint8(red.*tmp_blob);
g = uint8(green.*tmp_blob);
b = uint8(blue.*tmp_blob);



return;
```

```
%-----------------------------------------

function x = thresh(x, th, val)

[row col] = size(x);
for i=1:row
    for j=1:col
        if x(i,j) >= th(1) && x(i, j) <= th(2)
            x(i, j) = val;
        else
            x(i, j) = 0;
        end
    end
end

return;

%-----------------------------------------


function [r, g, b, n]  = find_blobs(BIN, type)
% This block supports the Embedded MATLAB subset.
% See the help menu for details.

idvec = zeros(1, 50);

[ROW COL] = size(BIN);

%{
if ROW>120 || COL>160
    r = uint8(BIN);
    g = uint8(BIN);
    b = uint8(BIN);
    n = -1;
    return;
end
%}

dwnsmple = 4;
```

```
row = round(ROW/dwnsmple);
col = round(COL/dwnsmple);
bin = uint8(zeros(row, col));

rw = 0;
% for i=dwnsmple+1:dwnsmple:ROW
for i=1:dwnsmple:ROW
    rw = rw + 1;
    co = 0;
    for j=1:dwnsmple:COL
        co = co + 1;
        bin(rw, co) = uint8(BIN(i, j));
    end
end

% r = uint8(zeros(row, col));
% g = uint8(zeros(row, col));
% b = uint8(zeros(row, col));
r = uint8(bin);
g = uint8(bin);
b = uint8(bin);
n = -1;%count;

% Bin = double(bin)/255;
% binsum = sum(Bin(:));
% ratio = binsum/(row*col);

% if ratio<=0.1
    % Find the 4 or 8 connected reachability matrix
    [eqtable, limg, maxlabel] = find_reachability_matrix(bin
       ↪ , type);
    if maxlabel>=1
        % Find equivalent labels
        table1 = updatetable(eqtable, maxlabel);
        table2 = updatetable2(table1, maxlabel);
        table3 = double(or(table1, table2));
        [idvec, n] = updatetable3(table3, maxlabel);
%
%
```

```
%           % Label the final image
          Limg = label_image(limg, n, idvec, maxlabel);
          r = uint8(Limg);
          g = uint8(Limg);
          b = uint8(Limg);
     end
%        r = uint8(limg);
%        g = uint8(limg);
%        b = uint8(limg);
% end




return;
%-------------------------------------------
function [eqtable, limg, maxlabel] =
   ↪ find_reachability_matrix(img, Type)

[row col] = size(img);

type = Type;
if Type~=4 && Type~=8
     type = 8;
end

% Step 1: Label 8 or 4 connected components
limg = zeros(row, col);
label = 0;
Maxlabel = 50;
for i=1:row
     for j=1:col
          if img(i, j)>0 && limg(i, j)==0
               if (j-1)>=1 && img(i, j-1)>0
                    limg(i, j)= limg(i, j-1);
               else
                    if label<=(Maxlabel-1)
                         label = label+1;
                         limg(i, j)= label;
                    else
                    limg(i, j)= 0;
                end
            end
        end
    end
end

% Find the reachability matrix
maxlabel = label;
eqtable = eye(Maxlabel, Maxlabel);
if maxlabel<1
    return;
end
% eqtable = eye(maxlabel, maxlabel);
for i=2:row
    for j=1:col
        if type == 4
            if img(i, j)>0
                if img(i-1, j)>0 && limg(i, j)~=0
                    eqtable(limg(i, j), limg(i-1, j)) = 1;
                    eqtable(limg(i-1, j), limg(i, j)) = 1;
                end
            end
        elseif type == 8
            if img(i, j)>0 && limg(i, j)~=0
                if (j-1)>=1 && img(i-1, j-1)>0
                    eqtable(limg(i, j), limg(i-1, j-1)) = 1;
                    eqtable(limg(i-1, j-1), limg(i, j)) = 1;
                end
                if img(i-1, j)>0
                    eqtable(limg(i, j), limg(i-1, j)) = 1;
                    eqtable(limg(i-1, j), limg(i, j)) = 1;
                end
                if (j+1)<=col && img(i-1, j+1)>0
                    eqtable(limg(i, j), limg(i-1, j+1)) = 1;
                    eqtable(limg(i-1, j+1), limg(i, j)) = 1;
                end
            end
        end
    end
end

return;
%-----------------------------------------------------
function [table] = updatetable(eqtable, maxlabel)

[row col] = size(eqtable);
table = zeros(row, col);
% etable = eqtable;
for i=1:maxlabel
    tmp1 = eqtable(i, :);
    for j=i+1:maxlabel
        tmp2 = eqtable(j, :);
        iscommon = sum(double(and(tmp1, tmp2)));
        if iscommon
            tmp1 = double(or(tmp1, tmp2));
        end
    end
        table(i, :) = tmp1;
end

return;
%-----------------------------------------------------
function [table] = updatetable2(eqtable, maxlabel)

[row col] = size(eqtable);
table = zeros(row, col);
% etable = eqtable;
for i=maxlabel:-1:1
    tmp1 = eqtable(i, :);
%          i
%          count = count + 1;
%          idvec(i) = count;
    for j=i-1:-1:1
        tmp2 = eqtable(j, :);
        iscommon = sum(double(and(tmp1, tmp2)));
        if iscommon
```

```
                tmp1 = double(or(tmp1, tmp2));
%                       find(tmp1==1)
%                       pause
          end
     end
     table(i, :) = tmp1;
end

return;
%------------------------------------------------
function [idvec, count] = updatetable3(eqtable, maxlabel)

[row col] = size(eqtable);
table = zeros(row, col);
count = 0;
idvec = zeros(1, col);
% etable = eqtable;
for i=1:maxlabel
     tmp1 = eqtable(i, :);
     if tmp1(i)~=0
          count = count + 1;
          idvec(i) = count;
          for j=i+1:maxlabel
               tmp2 = eqtable(j, :);
               iscommon = sum(double(and(tmp1, tmp2)));
               if iscommon
                    tmp1 = double(or(tmp1, tmp2));
               end
          end
          table(i, :) = tmp1;
          for j=i+1:maxlabel
               if tmp1(j)==1
                    idvec(j) = count;
                    eqtable(j, :) = table(j, :);
               end
          end
     end
end
```

```
return;
%-------------------------------------------------------
function [Limg] = label_image(limg, count, idvec, maxlabel)

[row col] = size(limg);
Limg = limg;
for i=1:row
    for j=1:col
        if Limg(i, j)>0 && Limg(i, j)<=maxlabel
%              if count<=255
%                  Limg(i, j) = uint8(round((255/count)*idvec
   ↪ (Limg(i, j))));
%              else
                 Limg(i, j) = uint8(idvec(Limg(i, j)));
%              end
        end
    end
end

return;
%-------------------------------------------------------



function [X_out, Y_out, maxN] = find_centroid(image, n, bin
   ↪ )
% This block supports the Embedded MATLAB subset.
% See the help menu for details.

[binrow bincol] = size(bin);
cr = bin;
cg = bin;
cb = bin;
[row col] = size(image);
x = -1;
y = -1;
maxN = -1;
if n>50
    X_out = -1;
    Y_out = -1;
    return;
end
X = zeros(1, 50);
Y = zeros(1, 50);
N = zeros(1, 50);
for L=1:n
    for i=1:row
        for j=1:col
            if image(i, j) == L
                X(L) = X(L) + j;
                Y(L) = Y(L) + i;
                N(L) = N(L) + 1;
            end
        end
    end
end

%maxN = -1;
maxNid = -1;
for L=1:n
    val = N(L);
    if val > maxN
        maxN = val;
        maxNid = L;
    end
end

if maxN > 0
    x = 4*(X(maxNid)/N(maxNid));
    y = 4*(Y(maxNid)/N(maxNid));

    for i=fix(y-2):fix(y+2)
        if i>=1 && i<=binrow
            for j=fix(x-2):fix(x+2)
                if j>=1 && j<=bincol
                    cr(i, j) = 255;
                    cg(i, j) = 0;
                    cb(i, j) = 0;
                end
            end
        end
    end

end

X_out = x;
Y_out = y;

return;
%----------------------------------------



% returns a ratio between object dimensions inferred from
   ↪ information given
% by the RGB and IR images.  ratio is between 0 and 1.0
function [X_out, Y_out, Acc_out, flag_out] =
   ↪ estimate_accuracy(X,Y,centroid_strength,r_cht, flag_in
   ↪ )
%#codegen

if flag_in == true
    flag_out = false;
    X_out = -1;
    Y_out = -1;
    Acc_out = -1;
else

    % Ignoring distortion and simplifying the factors of
       ↪ focal length and
    % aspect ratio to some constant K that will be
       ↪ determined experimentally...
    %
    % u = Kx*X/Z + Ox                (pinhole camera model eq
       ↪ for column position)
    % r_px = u1 - u2 = Kx*(X1-X2)/Z + Ox - Ox
    % X1-X2 = r_actual
    %
    % r_px = Kx*r_actual/Z
```

```
R_ACTUAL_MM = 63.5;      % website listed diameter as 5
    ↪ inches
Kx = 463.582;

[count, ~] = size(X);
Accuracy = double(zeros(size(X)));

for i=1:count
    if (centroid_strength(i) == 0 || X(i) == 0)
        Accuracy(i) = 0.0;
    else

        r_px = round((R_ACTUAL_MM * Kx) / X(i));    %
            ↪ infer approximate radius of projected
            ↪ object (in pixels)

        if r_cht == 0
            % Blob Detection used

            area_px = pi * r_px(i) * r_px(i);   %
                ↪ approximate image projection area

            % 0 <= E <= 1.0
            if (area_px > centroid_strength(i))
                Accuracy(i) = centroid_strength(i) /
                    ↪ area_px;
            else
                Accuracy(i) = area_px /
                    ↪ centroid_strength(i);
            end
        else
            % CHT used

            if (r_px > r_cht)
                Accuracy(i) = r_cht / r_px;
                else
                    Accuracy(i) = r_px / r_cht;
                end
            end
        end
    end



    % select by max acc instead...
    X_out = X(1);
    Y_out = Y(1);
    %Acc_out = Accuracy(1);
    Acc_out = centroid_strength;
    flag_out = true;
end



end %end function

function [Vr,Vl] = rotate()
%#codegen
scale = 0.05;

% can only turn left
Vr = scale;
Vl = -scale;



function [Vc,Wc] = UxUy2VcWc(tx, ty)
Max_Kv = .2
Max_Kw = .2

l=.0235; %#length between the two wheels in meters

FOLLOW_DISTANCE = .5;
K_W = 5;
K_V = .2;

distance = sqrt(tx^2 + ty^2) - FOLLOW_DISTANCE;
```

```
angle = atan2(ty,tx);


Wc = -K_W * angle;

Vc = K_V * distance;

if Vc > Max_Kv
    Vc = Max_Kv;
end


if Vc < -Max_Kv
    Vc = -Max_Kv;
end


function [Vr,Vl] = VcWc2VrVl(Vc,Wc)
%#codegen

l=.0235;
Vr = Vc+(1/2)*(l*2)*Wc;
Vl = Vc-(1/2)*(l*2)*Wc;


function [Px,Py] = PxPy_convert(x, y, theta)
%#codegen

l=.0235; %#length between the two wheels in meters
%R=.0175; %#R or signed distance from ICC (Instanteous
    ↪ center of curvature) in meters
Px= x+l*cos(theta)
Py = y+l*sin(theta)


function [Ux, Uy] = UxUy(Px,Py,Pxt,Pyt)
% pxt, pyt are the target's x and y
```

```
%#codegen
%rho_star=.5;
%omega_star=.5;

krho = .25/2; % forward v gain
kphi = .5/2;    %
w_d= 0.1;

U = [0 0];
rho_size = 1;
phi_size = 2;
[phi, rho] = cart2pol(Px,Py);
[phi_d, rho_d] = cart2pol(Pxt,Pyt);

[phi_dif, rho_dif] = cart2pol((Px-Pxt),(Py-Pyt));

PxD = Px - Pxt;
PyD = Py - Pyt;




J_i=[PxD/(sqrt(PxD^2+PyD^2)) PyD/(sqrt(PxD^2+PyD^2)); -PyD/(
    ↪ PxD^2+PyD^2) PxD/(PxD^2+PyD^2)];






%J_i=[(Px-Pxt)/phi_dif (Py-Pyt)/phi_dif;...
%     -(Py-Pyt)/((1+((Py-Pyt)/(PD))^2)*(Px-Pxt)^2)   1/((1+((
    ↪ Py-Pyt)/(PD))^2)*(Px-Pxt))];




%J_i=[ (Px-Pxt)/phi (Py-Pyt)/phi_dif;     -(Py-Pyt)/(phi_dif)
    ↪ ^2  (Px-Pxt)/(phi_dif)^2 ];

rhoi_dot = krho*(rho_size-rho_dif);
%phii_d= phi/2
%phii_d = phi_d - phii_d
```

```
phii_dot = w_d; %+ kphi*((phi_size-phi)/2);
V_i= [rhoi_dot; phii_dot];
Rt = [ 1 0; 0 1];



%U = Rt*(J_i'* V_i);
U = inv(J_i)*V_i;
Ux = U(1);
Uy = U(2);

function [Vc,Wc] = UxUy2VcWc(Ux, Uy, theta)
VC_MAX = 0.2;
WC_MAX = 0.1;
l=.0235; %#length between the two wheels in meters

kin_inv = inv([cos(theta) -l*sin(theta) ; sin(theta) l*cos(
    ↪ theta)]);
P = kin_inv*[Ux;Uy];
Vc = P(1);
Wc = P(2);
%
% if P(1) > VC_MAX
%       Vc = VC_MAX;
% else
%       Vc=P(1);
% end
%
% if P(2) > WC_MAX
%       Wc = WC_MAX;
% else
%       Wc = P(2);
% end

function [Vr,Vl] = VcWc2VrVl(Vc,Wc)
%#codegen

l=.0235;
Vr = Vc+(1/2)*(l*2)*Wc;
Vl = Vc-(1/2)*(l*2)*Wc;
```

```
function [Vr,Vl] = fcn(rot_en, enc_en, Vr_rotate,Vl_rotate,
    ↪ Vr_enc, Vl_enc )
%#codegen
if rot_en == 1
    Vr = Vr_rotate;
    Vl = Vl_rotate;
elseif enc_en == 1
    Vr = Vr_enc;
    Vl = Vl_enc;
else
   Vr = 0;
   Vl = 0;
end

function [vC,Omega] = fcn(vL,vR)
%#codegen

d= 0.235;

vC=0.5*(vL+vR);
Omega=(1/d)*(vR-vL);

function [x_dot,y_dot,psi_dot] = fcn(vC,Omega,psi)
%#codegen



x_dot=cos(psi)*vC;
y_dot=sin(psi)*vC;
psi_dot= Omega;
```

Appendix B

Appendix 2

B.1 Figures

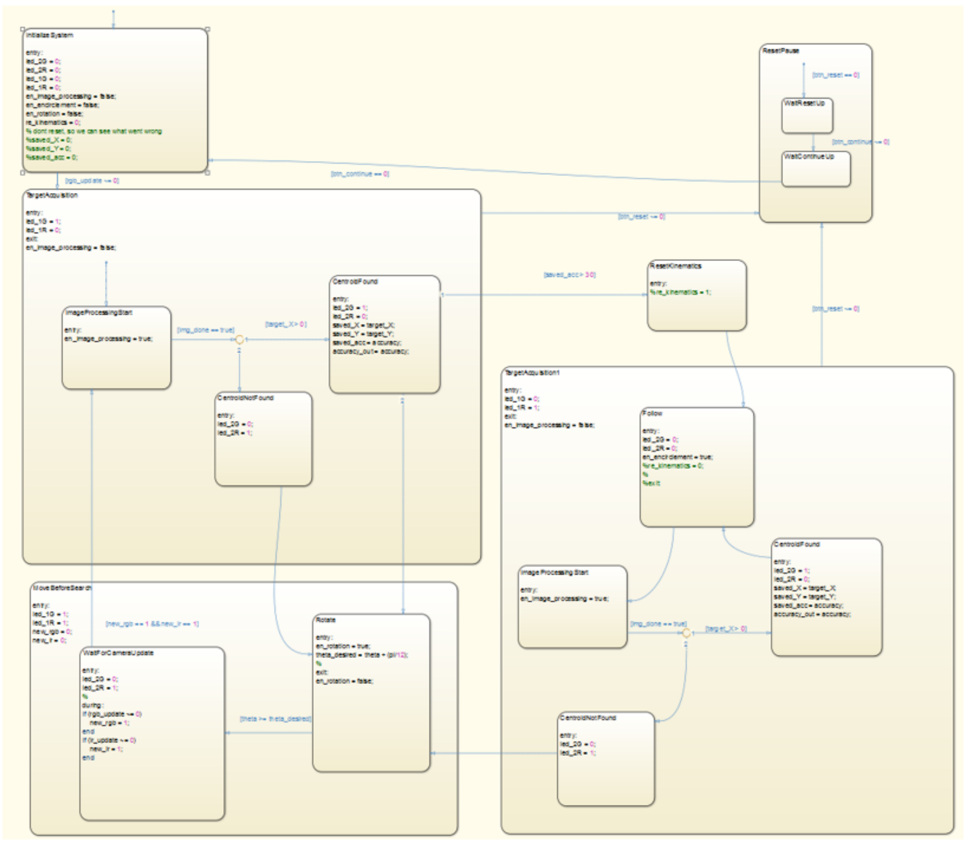
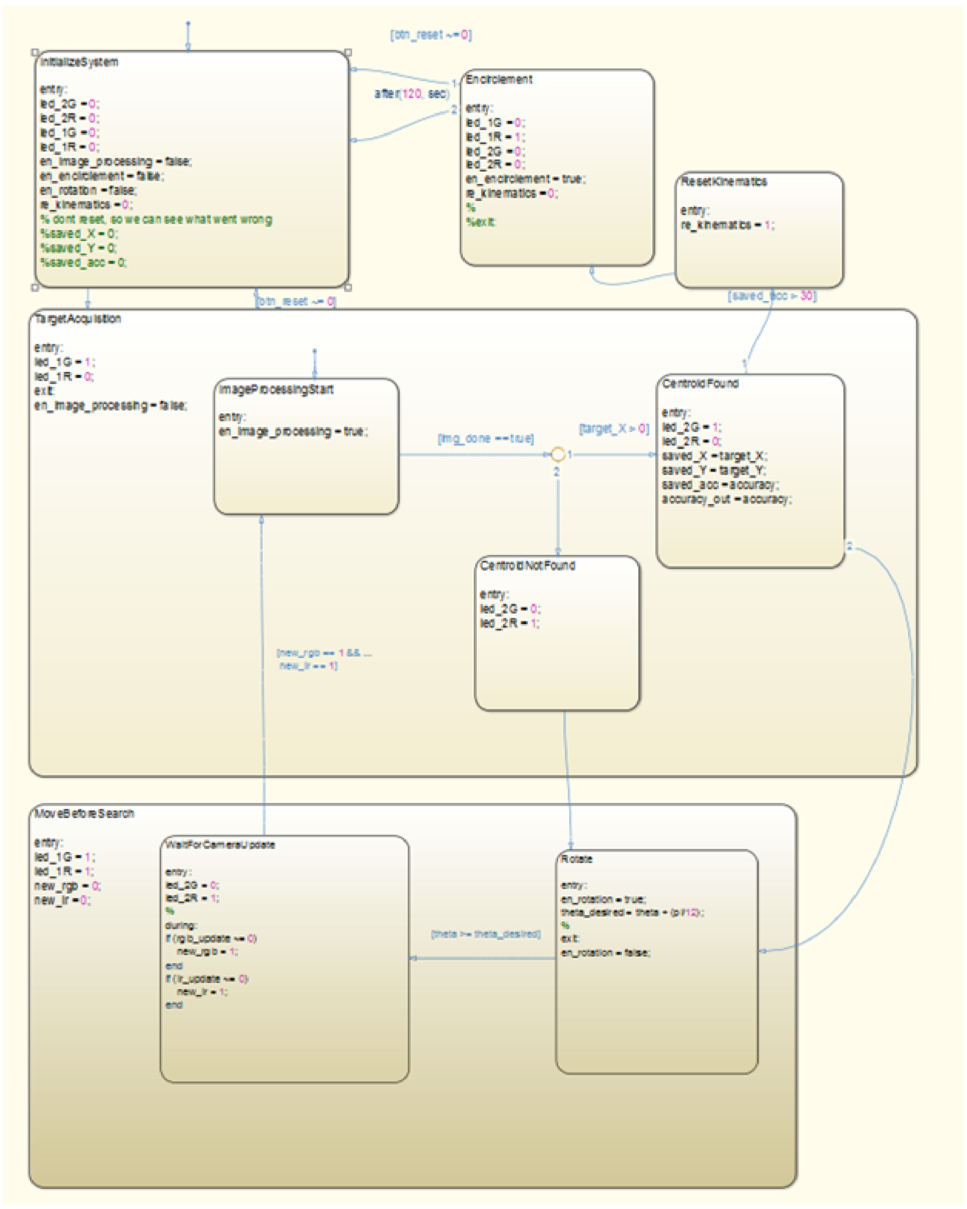


Figure B.1: Event-based System Control: Encirclement Control



```
InitializeSystem

entry:
led_2G = 0;
led_2R = 0;
led_1G = 0;
led_1R = 0;
en_image_processing = false;
en_encirclement = false;
en_rotation = false;
re_kinematics = 0;
% dont reset, so we can see what went wrong
%saved_X = 0;
%saved_Y = 0;
%saved_acc = 0;
```

```
Encirclement

entry:
led_1G = 0;
led_1R = 1;
led_2G = 0;
led_2R = 0;
en_encirclement = true;
re_kinematics = 0;
%
%exit:
```

```
ResetKinematics

entry:
re_kinematics = 1;
```

```
TargetAcquisition

entry:
led_1G = 1;
led_1R = 0;
exit:
en_image_processing = false;
```

```
ImageProcessingStart

entry:
en_image_processing = true;
```

```
CentroidFound

entry:
led_2G = 1;
led_2R = 0;
saved_X = target_X;
saved_Y = target_Y;
saved_acc = accuracy;
accuracy_out = accuracy;
```

```
CentroidNotFound

entry:
led_2G = 0;
led_2R = 1;
```

```
MoveBeforeSearch

entry:
led_1G = 1;
led_1R = 1;
new_rgb = 0;
new_ir = 0;
```

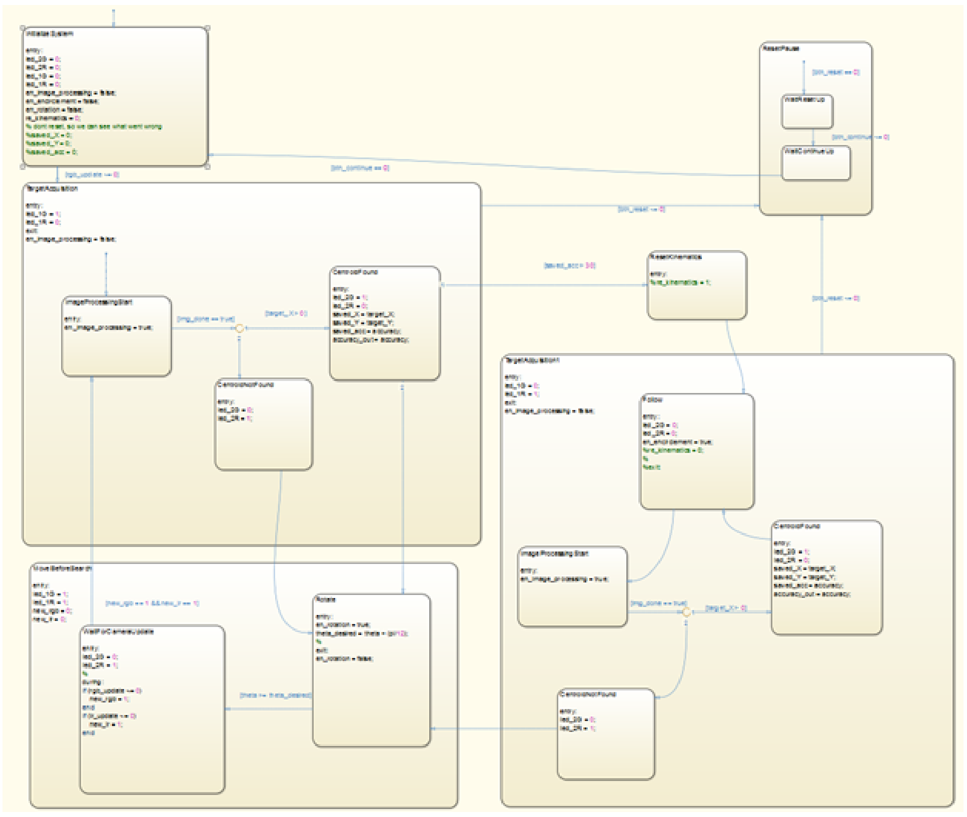
```
WaitForCameraUpdate

entry:
led_2G = 0;
led_2R = 1;
%
during:
if (rgb_update ~= 0)
   new_rgb = 1;
end
if (ir_update ~= 0)
   new_ir = 1;
end
```

```
Rotate

entry:
en_rotation = true;
theta_desired = theta + (pi/12);
%
exit:
en_rotation = false;
```

Figure B.2: Event-based System Control: Encirclement Control



```
InitializeSystem
entry:
led_2G = 0;
led_2R = 0;
led_1G = 0;
led_1R = 0;
en_image_processing = false;
en_encirclement = false;
en_rotation = false;
re_kinematics = 0;
% dont reset, so we can see what went wrong
%saved_X = 0;
%saved_Y = 0;
%saved_acc = 0;
```

```
TargetAcquisition
entry:
led_1G = 1;
led_1R = 0;
exit:
en_image_processing = false;
```

```
ImageProcessingStart
entry:
en_image_processing = true;
```

```
CentroidFound
entry:
led_2G = 1;
led_2R = 0;
saved_X = target_X;
saved_Y = target_Y;
saved_acc = accuracy;
accuracy_out = accuracy;
```

```
CentroidNotFound
entry:
led_2G = 0;
led_2R = 1;
```

```
ResetKinematics
entry:
%re_kinematics = 1;
```

```
TargetAcquisition1
entry:
led_1G = 0;
led_1R = 1;
exit:
en_image_processing = false;
```

```
Follow
entry:
led_2G = 0;
led_2R = 0;
en_encirclement = true;
%re_kinematics = 0;
%
%exit
```

```
Image Processing Start
entry:
en_image_processing = true;
```

```
CentroidFound
entry:
led_2G = 1;
led_2R = 0;
saved_X = target_X;
saved_Y = target_Y;
saved_acc = accuracy;
accuracy_out = accuracy;
```

```
CentroidNotFound
entry:
led_2G = 0;
led_2R = 1;
```

```
MoveBeforeSearch
entry:
led_1G = 1;
led_1R = 1;
new_rgb = 0;
new_ir = 0;
```

```
WaitForCameraUpdate
entry:
led_2G = 0;
led_2R = 1;
%
during:
if (rgb_update ~= 0)
   new_rgb = 1;
end
if (ir_update ~= 0)
   new_ir = 1;
end
```

```
Rotate
entry:
en_rotation = true;
theta_desired = theta + (pi/12);
%
exit:
en_rotation = false;
```

Figure B.3: Event-based System Control: Encirclement Control