Final Report

Click and Clack

Jason Monsorno
TAs: Mike Pridgen
Thomas Vermeer
Instructors: Dr. A. Antonio Arroyo
Dr. Eric M. Schwartz
University of Florida
Departments of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Design Laboratory

Table of Contents

Abstract 3
Executive Summary 3
Introduction 5
Integrated Systems 5
Mobile Platform 6
Actuation 6
Sensors 7
Behaviors 8
Experimental Layout and Results 9
Conclusion 9
Documentation 10
Appendices 10

Abstract

Click & Clack are cooperative retrieval robots. They will each use a camera to detect objects and depending on the color will retrieve the object itself or ask for help and both robots will simultaneously carry the object back to a storage bin.

Executive Summary

The project started as an idea of swarm robotics and turned into a smaller version of it. Click & Clack were cooperative in the individual retrieval of a certain color object, green for media day, using the exact same behaviors. After that the first robot started would take lead and retrieve any remaining objects of the secondary color which were presumed to be heavier. It put on a show to convey the idea of being heavier and to need help, and then the second robot would come to the rescue and help pick it up cooperatively. Though it ran to the wire in terms of completion time, the project was completed for media day.

Obtaining a base for the mobile platform started with my familiarity with the behavior and communication for the iRobot Create which is an educational and hobby release of the company’s popular Roomba just without the vacuum. After a few days of shopping online, three original series Roombas were purchased. The pros were starting with a fairly solid base that included bump sensors, motors, motor encoders, and wheel drop switches – the additional IR receiver and “cliff sensors” were removed for this project. The cons were that the original series did not have a serial interface like the later versions without soldering to the motherboard, had a different communication protocol, and had that pesky vacuum. The solution was to not use the serial communication but to interface with each component separately. The bump sensors and the encoders use break-beam infrared LED and photo-resistor sets so they took some resistors and then were hooked in directly. The motors were standard DC motors with attached capacitors which were plugged directly into a motor driver. The vacuum and all the brushes were removed which left plenty of room to work with but left the robot without a cover plate to properly access the space from the top.

Next the arm had to be made, to turn and be able to go up and down it had to have a gripping servos and at least 2 DOF. To maintain a uniform plane for the objects that were picked up, a third DOF was added at the “wrist” of the arm. This would ensure for pickup up and dropping off that the gripper/claw could be positioned at any reasonable angle while the distance away from center was controller by the servo on the same axis but lower on the arm. The last servo was used for rotating the entire arm which was considered a greater load and was designed to not use a standard two bearing system per joint so a heavier duty metal gear servo with internal bearings was used.

The robot required a way to identify the objects and the drop off point so a camera was the obvious choice. The camera used was a C3088 which is the same camera used on the CMU

camera system but used an onboard vision system built into the controller. The camera was mounted where the original infrared receiver was since it was in the front and centered, it nearly fit the camera lens perfectly so little was needed except for actually securing it down. The final assembly stages were motor drivers and wireless communication.

The controller had onboard motor driver but was limited to the base voltage of the controller which is not intended to exceed a 7.2v battery pack which did spin the motors but slowly and had a working range of 80% to 100% power otherwise would not spin. A motor driver was wired with 14.4v and used the onboard drivers as signals to drive the external motor driver. Everything seemed fine for about 2 weeks except for the day before demonstration day. The night before the robots were supposed to be complete and demoed, while working on the code to control an over tracking issue, the motor driver sparked a small fire and fried the motor driver, luckily nothing else was damaged. Since two robots were complete, the second robot's motor driver was taken for testing. About an hour later, that motor driver met the same fate with a proportionally large fire enough to melt the solder and remove the motor driver chip from the breakout board. After that and the unknown cause of their demise, the onboard slower motor driver seemed the logical choice especially considering time constraints.

The wireless communication was originally intended for implementation with some cheap RF communication chips, a separate frequency for each transmitter and receiver per robot. That did not work before demoing because no matter what, not a single byte – bit either for that matter – was received by the other board. Transmission and reception were examined using the oscilloscope and the controller board was sending bytes to the transmitter but nothing came out of the receiver. XBee modules were rushed shipped the day after with corresponding 5v to 3.3v adapters. They were hooked up as described directly to a rs232 signal but all the was received was gibberish, a byte received for every few bytes sent and the received was not even any of the bytes sent. After testing different baud rates and researching and testing a possible solution was discovered. Even though the XBee interface described excepting a rs232 signal and having the 5v adapter did reduce the signal for the XBee as specified necessary, the further documentation described how the byte was supposed to look, having a normally low signal. This meant the signal needed to be inverted so a 5v logic hex inverter chip, 7404, was used to invert the rs232 normally high signal. The result was receiving a byte for every one sent but it still being the wrong byte. After trying some conversion to see if that was possibly they issue, different baud rates were tried. Initially lower baud rates were tried to try and ensure no bytes were dropped and that the transistors, hex inverter, could fire fast enough, but the same results were produced. Finally a baud rate 9600 bps was set and every fell together, received byte was the sent byte.

Most of the basic behaviors and the communication were already coded but very little of the behaviors involving communication was tested since previously it was tested using a modified Ethernet cable as the serial connection. Very little modification was needed to the existed code but further behaviors for the secondary objects needed to be coded so that robots would visibly

help each other. That presented a few challenges since they always knew what each other were doing but did not actually know where they were, the simple solution to this was to use the drop off point and keep the robots positioned off that is at all possible. So when the total count of primarily objections was found, any robots dropping off objects would wait for the total count of dropped off object to also be the total count of primary object, then a secondary routine was started. Having then a primary robot, the other robots had to only find the primary robot or at least the object it brought back. Other than using the same color as the bin, this was not much of an issue and when along well.

The project did have quite a few hiccups that were at minimum resolved. Both robots did finish with identical builds and identical code as intended but did a different sequence of cooperative work. Additional aesthetics were added to the cooperative routine to appeal to the crowd which seemed to go over well and from peer response actually convinced some people in the crowd that the secondary objects were heavier. All in all, I was happy with the results.

Introduction

This project started because of my desire to make a practical robot. Making a robot follow a line was impressive for the technology years ago but is not practical for many real-world applications and has already been made a countless number of times. Swarm robotics is still a point of interest from the mechanical and programming aspects of robots as well as electrical which most fields seem to be. For my own sanity and wallet, the project will only include two robots but the environment is loosely design to allow more robots to coexist without disrupting the original two and will create more emergent behaviors.

Integrated Systems

Each robot is built identical so described the design and assembly of a single robot which can be reproduced. The robot main platform is an iRobot Roomba Original (fig 7.1) that has been modified. The cover, vacuum, and front infrared sensors have been removed. The original motor encoders, bump sensors, and wheel drop sensors. Added to the platform are infrared distance sensors, a camera, and a 3 DOF arm with claw for picking up objects in additional to the XBC v2 controller. The controller has two RF serial communication chips attached to communicate with the other robot. The motor encoders are hooked up directly to the motor output before going through a gearbox which increases the resolution. Each encoder uses the typical infrared photo-resistor break beam technology that most encoders utilize and are connected to digital ports. An internal clock monitors the encoder count and each motor speed change recalculates position. The bump sensors are also infrared photo-resistor break beams and are additionally connected to digital ports. Normally operation should not set off the bump sensors and are only monitored when tracking with the camera while the infrared distance sensors are ignored. The wheel drop sensors are physical switches attached to the suspension of each wheel and are monitored in a separate thread which will shutdown all other threads, motors, and servos. The infrared distance

sensors are homemade from SparkFun Electronics infrared LED and photo-resistor set (SKU SEN-00241) similar to the encoder and bump sensor. The set is put at an angle pointing towards each other with a separator. The output from the photo-resistor is nearly linear; therefore distance is calculated linearly. The camera is a 32-pin C3088 with an OmniVision OV6620 CMOS image sensor. The XBC controller has built-in video processor and comes with a camera library to interface with. The servo arm has 1 metal gear Towerpro MG946R for pan, 2 resin gear ElectriFly ES100 for same axis tilt, and another ES100 for the claw. Each servo is controlled by with 8-bit resolution for approximately 0.7 degree resolution. The controller is a XBC v2 – Xport Botball Controller version 2 – which utilizes a Nintendo GameBoy for coprocessing and display. The controller has 8 digital ports, 8 analog ports, 4 motor drivers and 4 servo ports in addition to the camera port and vision processing with blob tracking. Back EMF and vision functions are included in libraries. The RF communication was made using 7404 hex inverter chips and XBee modules, 1mW chip antenna, equipped with 5v adapters.

Mobile Platform

7.1

The platform for each robot is an iRobot Roomba Original Series (fig 7.1). With the complexity required of having multiple robots and the requirement of obstacle avoidance with a bump sensor, the Roomba family of robots was a good starting point. The Original Series was never equipped with the serial interface that all the later models had which iRobot has release the interface for so the platform has to be physical hacked to function with my controller board. Each sensor is fed directly into the controller board instead of the Roomba's motherboard. The motors had gone through a secondary motor driver for a 14.4v amplification to boost motor speed but were brought back down to 7.2v pack voltage.

Actuation

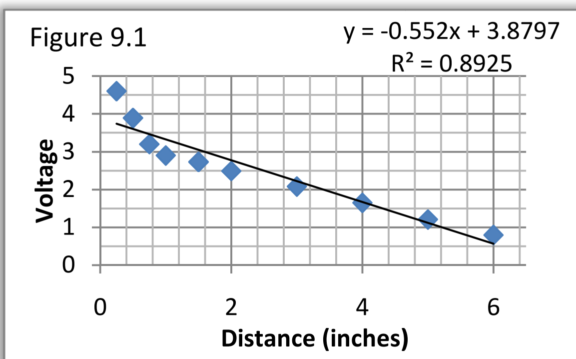
Each Roomba came equipped with a drive system which is comprised of a DC motor and a gearbox leading to the drive wheel. Each drive system is also hinged in the rear and has a spring to provide a simple form of shocks. A servo controller arm and grabber mechanism is added to the front to allow the robots to grab and lift objects. The panning servos is metal geared and includes bushings, the next servo is mounted directly to attached servo horn. The following two servos act as a single-axis tilt to allow the claw to be positioned at multiple distances with maintaining the same relative parallel angle to the ground to have consistent gripping action. Each of the two tilt servos has a secondary pivot point mounting on the opposite side of the servo in-line with the servo spline to act as a second bearing and to distribute the lateral load. The fourth servo is a mounted to the second tilt servo and is used for the claw assembly. The

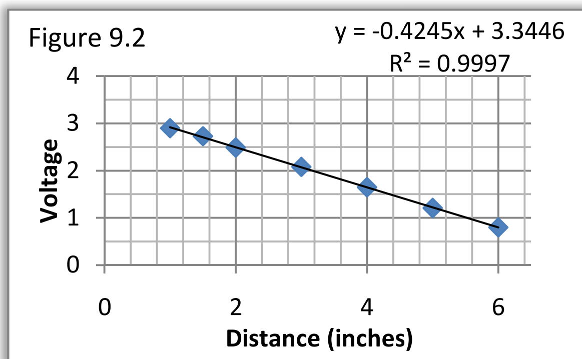
assembly has one sided motion. The entire arm and claw assembly is built from Lego pieces for easy prototyping changes and replication.

Sensors

Each robot is equipped with 2 infrared distance sensors, 4 infrared photo-resistor sets, 3 switches, and 1 camera. The infrared distance sensors are mounted in the front and are used for obstacle avoidance. Two infrared photo-resistor sets are used for bump detection and the other two are used as wheel encoders. Each wheel, 2 drive and 1 front caster, have limit switches on them to detect if the robot is not touching the ground with all the wheels. The camera is mounted to view the front of the robot for detecting and tracking objects.

The distance sensors give nearly linear output at distances from 1 to 6 inches as shown in Fig 9.1 and Fig 9.2. The controller supports 8-bit and 10-bit analog inputs. The 10-bit seems to





noticeable slower and giving random artifacts so 8-bit is used which is more than enough for basic obstacle avoidance. The bumper sensors and wheel encoders use the same technology of an infrared LED and photo-resistor set but are used with digital ports since the normal behavior uses digital logic and no further resolution is necessary. The encoders have a separate clock to monitor steps and are position is tracked upon each speed change to track encoder position with relative directions. Following each speed change of either or both motors, the encoders are read and reset. Using the robot wheel base, tire diameter, direction, and encoder step count, the relative position and orientation is calculated and the absolute – relative to starting origin – position and orientation are adjusted accordingly. The bump sensors are used when an object has been detected and is being approached, during this behavior the distance sensors are ignored to avoid false positives of obstacles as the intended object would be picked up and otherwise the distance sensor should be sufficient for obstacle avoidance by themselves. The three wheel drop switches are monitored by a separate thread and will shutdown all other threads if tripped and stop the motors and servos, this is used as a failsafe from the robot falling and so lifting the robot will stop execution.

The camera is controlled by built-in library functions. Feedback from three different color channels with blob tracking gives location and size of the object to track. The major flaw with

the built-in functions is the returned image is 356 x 292 while the largest blob size is limited to MAX_INT of 32767 which is 1/3 of the image's pixels; consequently, once the object's blob floods 1/3 of the screen, no further useful information is obtained while approaching the object. The major trick learned is to severely limit the color channel's color model to the extreme colors that stand out from background colors. Bright objects with the color models set in a small range of pixels around the brightest detected color will back the camera once recognize the top of the object and will provide extremely consistent data especially for round objects, balls. This has two major advantages, the first being that the camera is less likely to have false positives since the color model is smaller and unique to the desired object which if chosen correctly will be significantly different from common background colors of wall, chairs, etc. The second advantage is the blob size is relatively smaller to the entire object so it will not flood the camera's maximum size as soon providing more relative data and a closer range can be obtained virtually increasing distance resolution with the camera. A snippet of the camera functions from the html manual for the XBC in attached in the appendix.

Behaviors

The robots basic behavior is to go and pick up certain marked objects. The objects will either be orange or green so the camera can detect them. When a robot sees a green object then it will go retrieve it and brings it back to base. After all the green objects are picked up, when a robot sees an orange object, it asks for help from another robot. The other robot responds and object is picked up by both robots simultaneously. As a parameter, the total number of green and orange objects is set for the environment. The robots will communicate their finds to meet their combined goals of retrieving all the objects. Emergent behavior occurs when the robots find orange objects, when they are both trying to unload their retrieved object too close to each other, and when all the green objects are picked up if any orange objects are left.

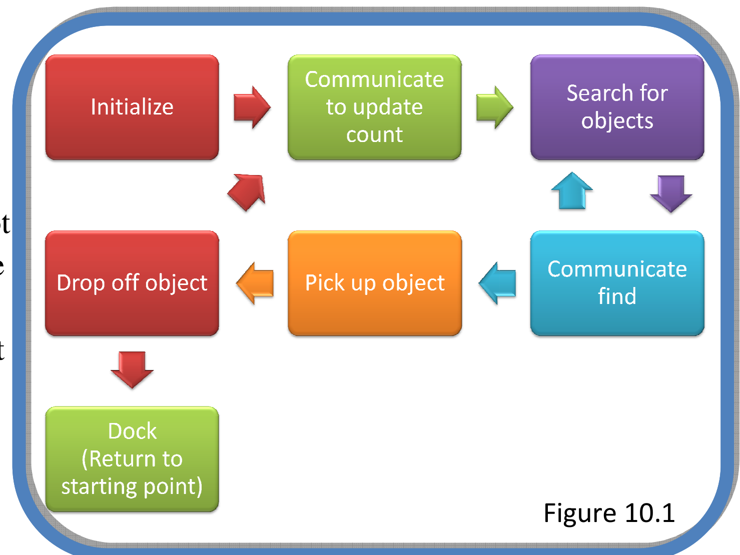


Figure 10.1

Fig 10.1 shows a basic block diagram of the behaviors and processes. Detailed behavior charts are included in the appendices.

Experimental Layout and Results

The layout will be a set starting point with a fixed relatively located base to drop to objects off in. The starting point will be surrounded by orange and green objects. The number of each is preset in the coding but is a defined constant and can be changed near instantly but requires a computer.

When more green balls were added, there was an occasion when the robots went after the same ball and even occasionally once the other robot picked it up it was still tracked. The least count possible to still show off the cooperative communication and waiting was planned to be ideal. For further testing and showing, the majority of the time only two were used.

Any number of orange balls worked since it was slight more straight-forward and only a single robot would retrieve them. Due to its theatrics and the overall slow speed, this was kept to a minimum as well but only for crowd appeal not for ideal behavior purposes.

The ideal numbers for combined behaviors and crowd appeal ended up being 2 green balls and 1 orange ball. With 2 green balls, each robot still had to retrieve a ball and one robot always had wait while the other was dropping their ball off which was a race condition with tracking the ball and then the bin which showed emergent behavior.

Conclusion

Both robots met the original specification, not as personally intended as though still fits the description. The key goals that were met were both robots have as close to identical build as possible, having the exact same code, and doing cooperative work. Each robot was built using the same components and wiring; other than minor measurements in controller placement and servo placement, the robots have identical builds. The code is 100% identical, no additional offsets or predetermined numbering of the robots was necessary. Both robots cooperate with each other during picking up the green balls and cooperatively pick up the orange balls meeting my goals.

Definitely the biggest backset was the wireless communication which was assumed to be more straight-forward than it was; mainly because of a lack of documentation with both wireless module attempts and ambiguity in serial protocol naming. The motor driver issue was definitely frustrating and cumbersome but luckily there was an alternative which though being slower that intended, was much easier to track with. The infrared distance sensors were on the robot and coded but they ended up physically breaking and had to be removed for media day, simple obstacle avoid was implemented with the bump sensors instead.

There is not much I would change with the robot. A longer and strong arm would be nice and possibly rerouting some of the cabling, primarily the camera cable. Otherwise the robots are fairly well off. I would have liked to cover it but from the appeal of using a GameBoy, I decided

against it. As far as the project goes, I should have tested the wireless communication earlier as well as done some basic coding for the encoders while I had some free time between assignments. I did not need all the data to make much of the logic and still referenced defined constants that were only necessary for actually running the robot; the math was the same and just included variables/constants.

This definitely was a challenge which was the reason I chose it and I would like to thanks to TA's for giving up the extra labs hours, their assistance in addition to being surrounded by peers just as frustrated helped me push on to complete this project through a few sleepless nights and countless other long days.

Documentation

Larry D. Moore, "File:Roomba original.jpg," *Wikipedia.org*, Feb. 27, 2006. [Online]. Available: http://en.wikipedia.org/wiki/File:Roomba_original.jpg.

KISS Institute, "XBC camera | Botball Store," *botballstore.org*, [Online]. Available: https://botballstore.org/content/xbc-camera.

KISS Institute, " XBC v2 Robot Controller | Botball Store," *botballstore.org*, [Online]. Available: https://botballstore.org/content/xbc-v2-robot-controller.

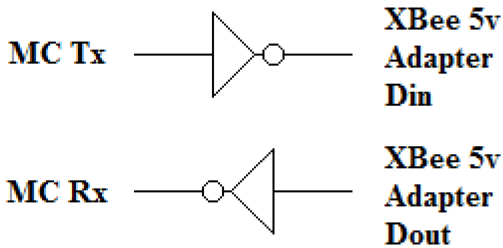
KISS Institute, "IC Vision API for the XBC," in *IC Programmers Manual*, Jan. 05, 2006 [Software Manual]. Available: http://botball.org/ic.

Appendices

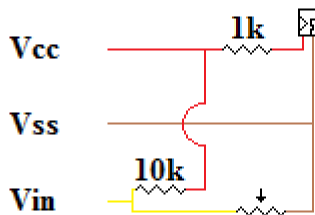
Appendix A – Figures pg 11
Appendix B – Flow charts pg 12
Appendix C – Manuals pg 14
Appendix D – Source code pg 21

Appendix A – Figures

RS232 normally high to normally low adapter logic diagram



Infrared LED photo-resistor set circuit diagram

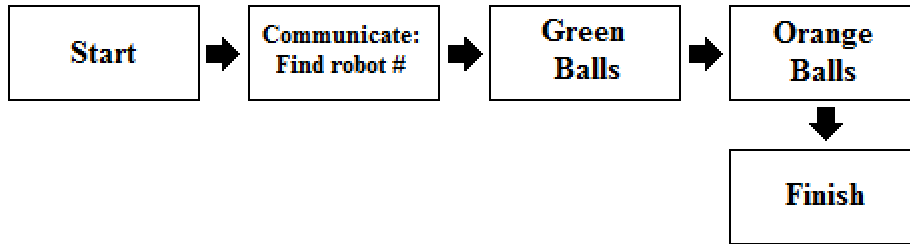


Communication protocols

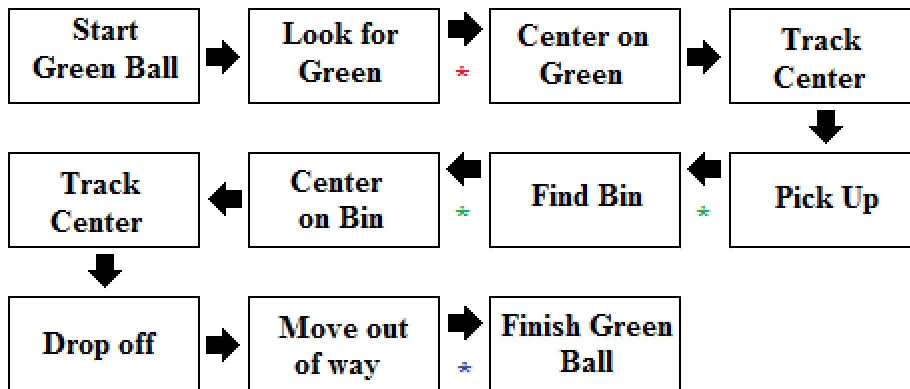
Function	**Event**	**Result**
Request Robot Number	At program start	Return other robot #+1 or defaults to 1
Ball Count	After each green ball is picked up	Other robot updates count
Ball Done Count	After each green ball is dropped off	Other robot updates done count
Base Busy	After bin align	Lock global resource
Base Free	After drop off	Unlock global resource
Request Help	After bin track, from primary	Request help from secondary
Helped	After orange ball grab, from secondary	Primary begins lift

Appendix B – Flow charts

Fundamental program process

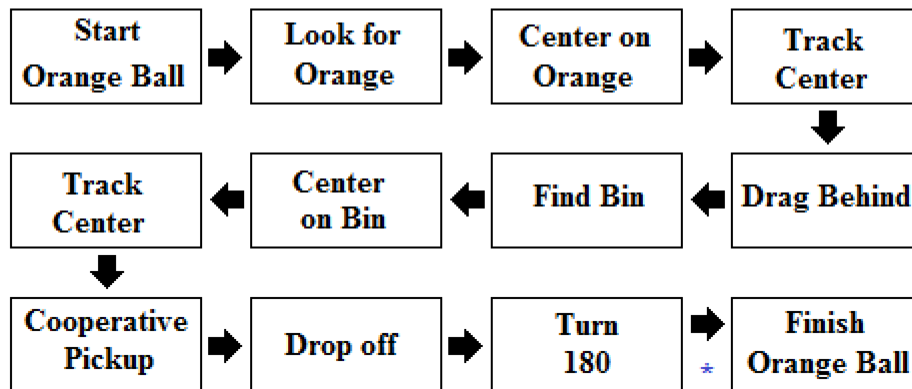


Green Ball procedure for both robots (Communication see Appendix A)



* "Look for Green" can be interrupted and exit the green ball behavior
* Will wait for base (bin) to be free before finding and before centering
* Condition depending on total ball count to start ball count, may loop to "Look for Green"

Orange Ball procedure for primary, Robot 1 (Communication see Appendix A)



* Condition depending on total ball count to start ball count, may loop to "Look for Green"

Appendix C – Manuals

XBC Manual (Camera Snippet)

IC Vision API for the XBC

- To use any camera routines, be sure to put
  `#use "xbccamlib.ic"`
  at the top of your file
- You must call
  `void init_camera();`
  to initialize the camera before any other camera functions will work

Tracking APIs

- Use
  `int track_is_new_data_available();`
  to determine if tracking data is available which is newer than the data processed by the last call to `track_update()`.
- Use
  `void track_update();`
  to process tracking data for a new frame and make it available for retrieval by the following calls.
- Use
  `long track_get_frame();`
  to return value is the frame number used to generate the tracking data.
- Use
  `int track_count(int ch);`
  to return the number of blobs available for the channel `ch`, which is a color channel numbered 0 through 2.
- Use the following functions of the form
  `int track_property(int ch, int i);`
  to return the value of a given property for the blob from channel `ch` (range 0-2), index `i` (range 0 to `track_count(ch)-1`). Fill in `track_property from one of the following:`

  - `track_size` gets the number of pixels in the blob, note that this maxes out (saturates) at 32,767 if the area gets that large
  - `track_x` gets the pixel x coordinate of the centroid of the blob
  - `track_y` gets the pixel y coordinate of the blob (note: 0,0 is the upper left; 356x292 is the lower right)
  - `track_confidence` gets the confidence for seeing the blob as a percentage of the blob pixel area/bounding box area

(range 0-100, low numbers bad, high numbers good)

- `track_bbox_left` gets the pixel x coordinate of the leftmost pixel in the blob
- `track_bbox_right` gets the pixel x coordinate of the rightmost pixel in the blob
- `track_bbox_top` gets the pixel y coordinate of the topmost pixel in the blob
- `track_bbox_bottom` gets the pixel y coordinate of the bottommost pixel in the blob
- `track_bbox_width` gets the pixel x width of the bounding box of the blob. This is equivalent to `track_bbox_right - track_bbox_left`
- `track_bbox_height` gets the pixel y height of the bounding box of the blob. This is equivalent to `track_bbox_bottom - track_bbox_top`

- Use
  void track_set_ch_enable(int ch, int val);
  int track_get_ch_enable(int ch);
  to enable or disable processing tracking data for a particular channel `ch` (range 0-2). The value passed into `val` or returned by track_get_ch_enable is 0=disabled, 1=enabled. All channels are enabled by default. Disabling unused channels is not required, but can increase performance.
- Use
  void track_enable_orientation();
  void track_disable_orientation();
  void track_set_orientation_enable(int val)();
  int track_orientation_enabled();
  to enable or disable orientation calculation, or get the current value of this setting (0=disabled, 1=enabled). This is disabled by default, and takes significant extra computation when enabled.
- When orientation calculation is enabled, use the following functions of the form
  float track_*property*(int ch, int i);
  to return the value of a given orientation-related property for the blob from channel `ch` (range 0-2), index `i` (range 0 to `track_count(ch)-1`). Fill in `track_property` from one of the following:

  - `track_angle` gets the angle in radians of the major axis of the blob. Zero is horizontal and when the left end is higher than the right end the angle will be positive. The range is -PI/2 to +PI/2.
  - `track_major_axis` gets the length in pixels of the major and minor axes of the bounding ellipse

```
track_minor_axis
```

- Use
```
void track_set_minarea(int minarea);
int track_get_minarea();
```
to set or retrieve the minimum area of a blob necessary to consider it valid. Blobs with area below `minarea` are ignored, and only blobs with area above `minarea` are returned by the above calls. Default value of min area is 100. The min area can be set interactively using the vision menus as well.
- Use
```
void track_show_display(int show_processed, int frameskip,
int channel_mask);
```
to show tracking results on the Gameboy display.
    - `show_processed` controls what type of video is displayed. If it is zero then raw video will be displayed, meaning that the video will be shown as it comes from the camera; if it is non-zero then processed video will be shown, meaning that pixels matching each color channel will be shown as a different color, and pixels matching no color channel will be shown as black.
    - `frameskip` controls how many frames of video are skipped between display updates. Smaller numbers will result in smoother video, but will heavily load the system and cause other computation to happen more slowly. Larger numbers will result in jerkier video, but allow other computaion more time to execute.
    - `channel_mask` controls which channels blob tracking data and/or processed video is shown for. The LSB controls channel 0, the next most significant bit controls channel 1, etc. A 1 in each bit position means to show that channel, and a 0 means to not show it. For example, 7 (0b111) shows all channels, 4 (0b100) shows just channel 2, etc.

A user may exit out of tracking display mode by hitting the B button on the Gameboy, which is consistent with the way display modes are exited when using the menu system.

Camera Configuration APIs

- Concepts:
    - *White Balance* refers to the "color temperature" the camera uses in converting the incoming light into pixel values. This is necessary because different light sources can contain a significantly different balance of red and blue components. For example, the sun and incandescent lights are much redder, and fluorescent lights are much bluer. Human brains compensate for changes in lighting color very quickly, to the point where we are mostly unaware that the issue exists.

      Cameras, however, need to use explicit mechanisms to try to compensate for these changes so that things in the image look "right" to humans, and more importantly for color segmentation, so that the HSV values reported by the camera when

looking at a given object can be fairly uniform when seen in a range of different lighting sources.

By default cameras turn on *Auto White Balance* (*AWB*) and dynamically adjust their color temperature to keep the amounts of red and blue in the field of view roughly balanced. This is fine if the content of the field of view **is** roughly balanced between red and blue and if the goal is to look good to humans in changing, arbitrary lighting situations. However, when trying to do color tracking, dynamically changing color temperature is generally counter productive.

Instead, you should use the Vision/Camera Config menu and press the Start button to interactively calibrate the white balance while pointing the camera at a white sheet of paper. This will adjust the *Red* and *Blue* components of the color temperature until the amount of each in the scene balances, then turn off *AWB* to lock those values in. After this procedure, it's a good idea to go to the Vision/Flash Memory menu, select "Setting: < Camera Config >", and then "Save to Flash".

- *Exposure* (range 0-154) refers to the amount of time during each frame that the camera spends allowing light to be detected. If the light is very bright this will be a small amount of time, since in bright light it doesn't take long to accumulate all the light the camera's detector can handle. If the light is dim then this will be a larger amount of time. By default the camera enables *Automatic Exposure Control* (*AEC*) and dynamically adjusts this value to maintain a constant relative percentage of "bright" and "dark" pixels. If you disable *AEC* then the *Exposure* will stay at whatever value it was last set to until *AEC* is enabled again.
- *Gain* (range 0-248) controls how much the raw image integrated from the incoming light is multiplied in order to generate the pixel values reported by the camera. If the light is bright enough, *Gain* should be zero. When the light is not bright enough, *Gain* has to be higher in order to compensate or the image will be too dark to be useful, but the quality of the image goes down and looks grainier. By default the camera enables *Automatic Gain Control* (*AGC*) and dynamically adjusts this value to maintain a constant relative percentage of "bright" and "dark" pixels. If you disable *AGC* then the *Gain* will stay at whatever value it was last set to until *AGC* is enabled again.
- Exposure/Gain xpoSetting:
    - The camera will report the *Exposure* and *Gain* values which it is currently using independent of whether *AEC* or *AGC* are enabled.
    - The user can directly set the *Gain* value (which also implicitly disables *AGC*), but the camera does not support not support directly setting the *Exposure* value.
    - There are two parameters which allow the user to adjust the way the camera dynamically adjusts its *Exposure* and *Gain*:
        - *Auto Exposure Ratio* (*AERatio*) (range 1- 254, default=65) controls the percentage of "bright" versus "dark" pixels which it tries to

maintain: 1 = Maintain 0.5% "bright" pixels, 65 = 25% "bright", 254 = 99.5% "bright". The net effect of this is that low AERatio values make the image look darker, and high values make the image look brighter.

- *Exposure Reference Level* (*ExpRL*) (range 0-224, default 160) selects the reference level voltage used for automatic setting of *Exposure* and *Gain*. Higher values make the image look brighter, and lower values make it look darker. This is actually a 3-bit value in the most significant 3 bits of a byte, so value changes are in increments of 32.

- When *AEC* and *AGC* are both enabled, the camera will set *Gain* to zero if the light is bright enough and modify *Exposure* to achieve the desired percentage of "bright" pixels given the current values of *AERatio* and *ExpRL*. As the light level decreases the camera will increase *Exposure* until it hits the maximum value (154), then modify *Gain* as much as it needs to to achieve its goals or until it hits the maximum value (248).

- Use
  int camera_get_awb();
  int camera_set_awb(int enable);
  to get or set whether or not *Auto White Balance* is enabled (0=disabled, 1=enabled).
- Use
  int camera_get_wb_color_temp(int color[]);
  int camera_set_wb_color_temp(int color[]);
  to get or set the red and blue components of color temperature. Calling camera_set_wb_color_temp implicitly disables *AWB*. `color[]` is an int array of length 2 where:
  - `color`

    `[0] = red`
  - `color`

    `[1] = blue`

  The return values are 0 for success, -1 for failure (fails if _array_size(color)!=2). If you want to use these functions from the interaction window you will need to use a block to create a `color[]` array:

  - {int color[2]; camera_get_wb_color_temp(color); printf("Red=%d, Blue=%d\n", color[0], color[1]);}
  - {int color[]={100,200}; camera_set_wb_color_temp(color);}
- The following are equivalent in function, but may be more convenient for interactive use:

  int camera_get_wb_red_temp();
  int camera_get_wb_blue_temp();

- Use
  int camera_get_aec();
  int camera_set_aec(int enable);
  to get or set whether or not *Auto Exposure Control* is enabled (0=disabled, 1=enabled).
- Use
  int camera_get_exposure();
  to get the current value of *Exposure*. There is no set function for *Exposure* because the camera does not support that operation.
- Use
  int camera_get_aec_ratio();
  int camera_set_aec_ratio(int val);
  to get or set the value of the *Auto Exposure Ratio* (*AERatio*) (range 1- 254, default=65).
- Use
  int camera_get_exp_ref_level();
  int camera_set_exp_ref_level(int val);
  to get or set the value of the *Exposure Reference Level* (*ExpRL*) (range 0-224, default 160).
- Use
  int camera_get_agc();
  int camera_set_agc(int enable);
  to get or set whether or not *Auto Gain Control* is enabled (0=disabled, 1=enabled).
- Use
  int camera_get_gain();
  int camera_set_gain(int val);
  to get or set the value of *Gain* (range 0-248).

  Calling camera_set_gain also implicitly disabled *AGC*.

Color Model APIs

- Color models are expressed as a range of values in the HSV (Hue, Saturation, Value) cube which are considered to be included within the range of pixels accepted by that model.
  - Hue (range 0-359) is analogous to what we usually think of as the "color" of a pixel: Red ~= 0, Green ~= 100, Blue ~= 240. The Hue range may wrap, for example *hMin*=340, *hMax*=10 is a valid range. However, the distance from *hMax* to *hMin* ((360 + *hMax* - *hMin*) % 360) may not exceed 120.
  - Saturation (range 0 - 223) is how pure and intense the hue is: 0 = totally unsaturated, such as black, white, or grey; 223 = totally saturated, such as neon orange, fire-engine red, etc. Hue is most reliable, and therefore color distinction is more robust, for pixels with high Saturation. If Saturation is too low, then the Hue calculation will be fairly random. Therefore you can set *sMin*, the minimum acceptable Saturation value, but *sMax* is the maximum possible value of 223 for color tracking.
  - Value (range 0-223) is how dark or bright the pixel is: 0 = black, 223 = bright. Hue is most reliable, and therefore color distinction is more robust, for pixels with

high Value. If Value is too low, then the Hue calculation will be fairly random. Therefore you can set *vMin*, the minimum acceptable Value, but *vMax* is the maximum possible value of 223 for color tracking.

- Use

```
int color_get_model(int model_num, int model[]);
int color_set_model(int model_num, int model[]);
```

  to get or set the parameters of color model `model_num` (range 0-2) currently in use. `model[]` is an int array of length 4 where:
    - `model[0] = hMin`
    - `model[1] = hMax`
    - `model[2] = sMin`
    - `model[3] = vMin`

  The return values are 0 for success, -1 for failure (fails if _array_size(model)!=4 or arguments out of range). If you want to use these functions from the interaction window you will need to use a block to create a `model[]` array:

    - {int model[4]; color_get_model(0, model); printf("H=(%d->%d), S>=%d, V>=%d\n", model[0], model[1], model[2], model[3]);}
    - {int model[]={0, 100, 200, 200}; color_set_model(0,model);}
- The following are equivalent in function, but may be more convenient for interactive use:

```
int color_get_ram_hmin(int model_num);
int color_get_ram_hmax(int model_num);
int color_get_ram_smin(int model_num);
int color_get_ram_smax(int model_num);
int color_get_ram_vmin(int model_num);
int color_get_ram_vmax(int model_num);
int color_set_ram_model(int model_num, int hmin, int hmax, int smin, int vmin);
```

Appendix D – Source Code

CandC.ic

```
/* * * * * * * * * * * * * * * * * * * * *

 * Title: Click and Clack              *
 * Author: Jason Monsorno              *
 * Language: Interactive C 8.0.2 for XBC *
 * Created: Feb. 28th, 2010            *
 * Last Modified: Apr. 19st, 2010       *
 *                            *
 * * * * * * * * * * * * * * * * * * * * */
#use "xbccamlib.ic"
#use "xbcserial.ic"
#use "globals.ic"
#use "functions.ic"
#use "calibration.ic"
#use "bump_avoidance.ic"
#use "mission.ic"

//Menu
#define options 5
char arrMenu[options][18] = {"Mission", "Bump Avoid", "Calibration", "Pinouts", "Serial
Reset"};

void main ()
{
   int selection = 0;
   int prevSelec = -1;
   int i;
   do
     {
      selection = 0;
      while (a_button());
      while(!a_button())//Menu

        {
         if(selection != prevSelec)
           {
            display_clear();
            for(i = 0; i < options; i++)
              {
               if (selection == i)
                 {
                  printf("* ");
                }
               else
```

```
            {
              printf("  ");
            }
            printf(arrMenu[i]);
            printf("\n");
          }
          prevSelec = selection;
        }
        if (down_button())
          {
          while(down_button()); //wait for release
          selection = min(selection + 1, options - 1);
        }
        if (up_button())
          {
          while(up_button()); //wait for release
          selection = max(selection - 1, 0);
        }
      }
      while(a_button());
      display_clear();
      printf("Starting \"");
      printf(arrMenu[selection]);
      printf("\"\n");
      switch(selection)
        {
        case 0:
          mission();
        break;
        case 1:
          bump_avoidance();
        break;
        case 2:
          calibration();
        break;
        case 3:
          pinouts();
        break;
        case 4:
          serial_set_mode(0);
        break;
        default:
          printf("Not yet coded, good-bye");
      }
  } while (selection >= 2);
}
```

Mission.ic

```
void mission()
{
  int moved2 = 0;
  BaseMid;
  ArmExtend;
  GripOpen;
  enable_servos();
  sleep(.75);
  ArmUp;
  initialize();
  serial_set_mode(BAUD_RATE);
  start_process(serial_thread());
  request_robot_number();
  sleep(1.0);
  if (robot_number == 0)
   robot_number = 1; //incase no response
  init_camera();

  printf("Robot %d starting\n", robot_number);
  NormalFind();
  if (global_ball_count_done < START_COUNT)
   {
    moved2 = 1;
    Turn(45L, robot_number-1);
    MoveSteps(8000L, -100);
    Turn(135L, robot_number-1);
    changeSpeed(0, 0);
    printf("Waiting on other robot\n");
  }
  while(global_ball_count_done < START_COUNT)
   msleep(100L);
  printf("Cooperative mode on\n");
  if(robot_number == 1)
   {
    if (!moved2)
     Turn(180L, dirC);
    SpecialFind();
  }
  else if (!moved2)
   {
    Turn(45L, robot_number-1);
    MoveSteps(8000L, -100);
    Turn(90L, robot_number-1);
    MoveSteps(2000L, -100);
    changeSpeed(0, 0);
```

```
  }
  else
  {
    Turn(45L, 1 - (robot_number-1));
    MoveSteps(2000L, -100);
    changeSpeed(0, 0);
  }
  if (robot_number == 2)
   {
    helpMode();
  }
  changeSpeed(0,0);
  beep();
  msleep(50L);
  beep();
  msleep(50L);
  beep();
  msleep(50L);
  beep();
  msleep(50L);
  beep();
  msleep(50L);
  disable_servos();

}

void helpMode()
{

  int intSpecial = 0;
  while (intSpecial < SPEC_COUNT)
   {
    while(!coop_help)
     msleep(100L);
    printf("Helping, hold on\n");
    search(SPECIAL, 5);
    if (track_size(SPECIAL, 0) > pickUpSize - 1000)
     MoveSteps(4000L, -100);
    align(SPECIAL);
    trackUntilSize(SPECIAL, pickUpSize);
    changeSpeed(0,0);
    ArmDown;
    GripOpen;
    sleep(.5);
    MoveSteps(750L, 100);
    changeSpeed(0,0);
```

```
        write_helped();
        GripClose;
        ArmExtend;
        sleep(1.75);
        GripOpen;
        sleep(.25);
        MoveSteps(1000L, -100);
        changeSpeed(0,0);
        coop_help = 0;
        intSpecial++;
    }
    Turn(45L, dirCC);
}

void SpecialFind()
{
    while (intSpecial < SPEC_COUNT)
      {
        if (DEBUG)
          printf("Starting Search\n");
        search(SPECIAL, 2);
        if (DEBUG)
          printf("Starting Pickup\n");
        if (pickup_special(SPECIAL))
          {
            intSpecial++;
            while(!dropoff(BIN));
            sleep(1.5);
            changeSpeed(0, 0);
            ArmFakeUp();
            beep();
            beep();
            beep();
            printf("HELP ME\n");
            write_request_help();
            while(!coop_help)
              msleep(10L);
            set_servo_position(ServoJoint, 190);
            sleep(.25);
            ArmExtend;
            sleep(2.25);
            ArmUp;
            BaseMid;
            sleep(.65);
            changeSpeed(100, 100);
            while(!SensBump);
```

```
      changeSpeed(0, 0);
      ArmDeploy;
      sleep(1.5);
      GripOpen;
      sleep(.5);
      ArmUp;
      if (DEBUG)
        printf("Count Remaining %d\n", absJ(SPEC_COUNT - intSpecial));
      Turn(180L, dirC);
    }
  }
}

void NormalFind()
{
  while (global_ball_count < START_COUNT)
   {
    if (DEBUG)
      printf("Starting Search\n");
    if (!search(NORMAL, 0))
      return;
    if (DEBUG)
      printf("Starting Pickup\n");
    if (pickup(NORMAL))
     {
      local_ball_count++;
      global_ball_count = local_ball_count + away_ball_count;
      write_ball_count();
      Turn(180L, dirC);
      changeSpeed(0,0);
      while(!dropoff(BIN));
      changeSpeed(100, 100);
      while(!SensBump);
      changeSpeed(0, 0);
      ArmDeploy;
      sleep(1.5);
      GripOpen;
      sleep(.5);
      ArmUp;
      write_base_free();
      local_ball_count++;
      write_ball_count_done();
      if (DEBUG)
        printf("Count Remaining %d\n", absJ(START_COUNT - global_ball_count));
    }
  }
```

```
}

void ArmFakeUp()
{
   int i;
   for(i = 140; i <= 170; i++)
     {
      set_servo_position(ServoJoint, i);
      msleep(30L);
    }
   sleep(.35);
   ArmDown;
   for(i = 140; i <= 190; i++)
     {
      set_servo_position(ServoJoint, i);
      msleep(30L);
    }
   sleep(.55);
   ArmDown;
   for(i = 110; i <= 140; i++)
     {
      set_servo_position(ServoArm, i);
      msleep(40L);
    }
   sleep(1.15);
   ArmDown;
}

int dropoff(int intChannel)
{
   if(base_busy)
     printf("Base is busy\n");
   while(base_busy);
   if (intSpecial > 0)
     search(intChannel, 6);
   else
     search(intChannel, 1);
   if(base_busy)
     changeSpeed(0,0);
   while(base_busy);
   write_base_busy();
   if (motorSpeedLeft == 0 && motorSpeedRight == 0)
     sleep(3.0); //Wait for other robot to move
   printf("Aligning with base\n");
   align(intChannel);
   printf("Tracking with base\n");
```

```
  return trackUntilSize(intChannel, dropOffSize);
}

int pickup_special(int intChannel)
{
  long delay = TIME_90;
  int pos = get_servo_position(ServoBase);
  if (!pickupPart1(intChannel))
   return 0;
  pickupPart2(intChannel);

  delay = TIME_90 / (long)(BaseRightPosition - BaseMidPosition) / 2L;
  changeSpeed(-MAX_SPEED, MAX_SPEED);
  while(pos != BaseRightPosition)
   {
    set_servo_position(ServoBase, pos++);
    msleep(delay);
  }
  return 1;
}

int pickupPart1(int intChannel)
{
  if (DEBUG)
   printf("Starting Align\n");
  write_base_free();
  align(intChannel);
  if (DEBUG)
   printf("Starting Track\n");
  if (!trackUntilSize(intChannel, pickUpSize))
   return 0;
  return 1;
}

void pickupPart2(int Channel)
{
  changeSpeed(0,0);
  GripOpen;
  sleep(.25);
  ArmDown;
  sleep(.5);
  changeSpeed(MAX_SPEED, MAX_SPEED);
  sleep(2.15);
  changeSpeed(0, 0);
  GripClose;
  sleep(.5);
```

```
}


int pickup(int intChannel)
{
   if (!pickupPart1(intChannel))
     return 0;
   pickupPart2(intChannel);
   ArmUp;
   sleep(1.5);
   return 1;
}

int trackUntilSize(int intChannel, int intSize)
{
   int x;
   track_update();
   while (track_size(intChannel, 0) < intSize && foundReliable(intChannel))
     {
      x = adjustX(intChannel);
      if (x < 0)
        changeSpeed(scaleSM(x, TRACK_FACTOR), MAX_SPEED);
      if (x >=0)
        changeSpeed(MAX_SPEED, scaleSM(x, TRACK_FACTOR));

      track_update();
    }
   return foundReliable(intChannel);
}

void align(int intChannel)
{
   int x, y;
   track_update();
   x = adjustX(intChannel);
   if (x < -25)
     {
      changeSpeed(-turnSpeed, turnSpeed);
      do
        {
         track_update();
         x = adjustX(intChannel);
      }while(x < -25);
    }
   if (x > 25)
     {
```

```
        changeSpeed(turnSpeed, -turnSpeed);
        do
          {
            track_update();
            x = adjustX(intChannel);
        }while(x > 25);
    }
}

int adjustX (int intChannel)
{
    return track_x(intChannel, 0) - (SCREEN_WIDTH / 2);
}

int adjustY (int intChannel)
{
    return track_y(intChannel, 0) - (SCREEN_HEIGHT / 2);
}

int search(int intChannel, int stage)
{
    printf("Searching Stage %d\n", stage);
    if (stage != 5 && (robot_number == 2 || stage == 2 || stage == 6))
      changeSpeed(-turnSpeed, turnSpeed);
    else
      changeSpeed(turnSpeed, -turnSpeed);
    track_update();
    while (!foundReliable(intChannel))
      {
        track_update();
        if (stage == 0 && local_ball_count + away_ball_count  >= START_COUNT)
          return 0;
    }
    return 1;
}

int foundReliable(int intChannel)
{
    return !(!(track_count(intChannel) > 0 && track_confidence(intChannel, 0) > 20 &&
track_size(intChannel,0) > 500));
}
```

Globals.ic

```
#define DEBUG 1

#define START_COUNT 2
#define ORANGE 0
#define GREEN 2
#define BLUE 1
#define SPEC_COUNT 1
#define NORMAL GREEN
#define SPECIAL ORANGE
#define BIN ORANGE

#define MAX_SPEED 100
#define ROTATION 6017L
#define TIME_90 3471L

#define SCREEN_WIDTH 356
#define SCREEN_HEIGHT 292

#define dirC    0
#define dirCC   1
#define dirRand 2

#define TRACK_FACTOR 1
#define turnSpeed    100

//Pin definitions
#define BumpLeft     15
#define BumpRight    8
#define DropFront    9
#define DropLeft     11
#define DropRight    10
#define EncoderLeft  13
#define EncoderRight 14
#define MotorLeft    3
#define MotorRight   1
#define ServoBase    0
#define ServoArm     1
#define ServoJoint   2
#define ServoGrip    3

//Servo functions
#define GripClose set_servo_position(ServoGrip, 50)
#define GripOpen  set_servo_position(ServoGrip, 128)
#define ArmUpPosition 245
#define JointUpPosition 10
```

```
#define ArmUp     {set_servo_position(ServoArm,   ArmUpPosition); sleep(.45);
set_servo_position(ServoJoint,  JointUpPosition);}
#define ArmDown   {set_servo_position(ServoJoint, 140); sleep(.25);
set_servo_position(ServoArm,   110);}
#define ArmExtend {set_servo_position(ServoArm,   170);
set_servo_position(ServoJoint, 100);}
#define ArmDeploy {set_servo_position(ServoArm,   200);
set_servo_position(ServoJoint,  50);}
#define BaseRightPosition 206
#define BaseLeftPosition 16
#define BaseMidPosition 110
#define BaseLeft  set_servo_position(ServoBase, BaseLeftPosition)
#define BaseMid   set_servo_position(ServoBase, BaseMidPosition)
#define BaseRight set_servo_position(ServoBase, BaseRightPosition)


//Sensor definitions
#define SensBL !digital(BumpLeft)
#define SensBR !digital(BumpRight)
#define SensBump (SensBL | SensBR)
#define SensDF digital(DropFront)
#define SensDL digital(DropLeft)
#define SensDR digital(DropRight)
#define SensDrop (SensDF | SensDL | SensDR)
#define SensEL read_encoder(EncoderLeft)
#define SensER read_encoder(EncoderRight)

//Used for drop sensor shutdown
int _shut_down_pid;
int global_x = 0, global_y = 0;

//Calibration
int pickUpSize = 19500;
int dropOffSize = 32767;

//Motor Tracking
int motorSpeedLeft = 0;
int motorSpeedRight = 0;

//Buttons
int arrButton[8] = {1, 2, 16, 32, 64, 128, 256, 512};
#define BUTTON_A 1
#define BUTTON_B 2
#define BUTTON_RIGHT 16
#define BUTTON_LEFT 32
#define BUTTON_UP 64
```

```
#define BUTTON_DOWN 128
#define BUTTON_R 256
#define BUTTON_L 512

//Ball Count
int base_busy = 0;
int global_ball_count = 0;
int local_ball_count = 0;
int away_ball_count = 0;
int global_ball_count_done = 0;
int local_ball_count_done = 0;
int away_ball_count_done = 0;

int robot_number = 0;
#define BAUD_4800 BAUD_9600*2
#define BAUD_2400 BAUD_4800*2
#define BAUD_1200 BAUD_2400*2
#define BAUD_RATE BAUD_9600

int coop = 0;
int intSpecial = 0;
int coop_help = 0;
int ball_coordinates[4] = {0, 0, 0, 0};
```

Calibration.ic

```
#define calOptions 6
char calArrMenu[calOptions][18] = {"Pickup Distance", "360 Rotation", "90 Test", "Serial
Read", "Serial Write", "Exit to Main Menu"};

void calibration()
{
  int selection = 0;
  int prevSelec = -1;
  int i;
  do
    {
     prevSelec = -1;
     selection = 0;
     while (a_button());
     //Menu
     while(!a_button())
       {
        if(selection != prevSelec)
          {
           display_clear();
           for(i = 0; i < calOptions; i++)
             {
              if (selection == i)
                {
                 printf("* ");
              }
              else
                {
                 printf("  ");
              }
              printf(calArrMenu[i]);
              printf("\n");
           }
           prevSelec = selection;
        }
        if (down_button())
          {
           while(down_button()); //wait for release
           selection = min(selection + 1, calOptions - 1);
        }
        if (up_button())
          {
           while(up_button()); //wait for release
           selection = max(selection - 1, 0);
        }
```

```
        }
        while(a_button());
        display_clear();
        //    printf("Starting \"");
        //    printf(arrMenu[selection]);
        //    printf("\"\n");
        switch(selection)
          {
            case 0:
              pickUpDistance();
            break;
            case 1:
              rotation();
            break;
            case 2:
              test90();
            break;
            case 3:
              serial_reader();
            break;
            case 4:
              serial_writer();
            break;
            case calOptions-1: //Exit to menu
              break;
            default:
              printf("Not yet coded, good-bye\n");
            sleep(1.0);
        }
    }while(selection != calOptions-1);
}

void serial_reader()
{
    int val;
    serial_set_mode(BAUD_RATE);
    while(1)
      if(serial_buffer_count() > 0)
        {
          val = serial_read_byte();
          printf("%x = %d\n", val, val);
       }

}
```

```
void serial_writer()
{
   int i=0;
   sleep(1.0);
   serial_set_mode(BAUD_RATE);
   serial_write_byte(201);
   serial_write_byte(0x0D);
   sleep(.75);
   serial_write_byte(202);
   serial_write_byte(0x0D);
   sleep(.75);
   serial_write_byte(203);
   serial_write_byte(0x0D);
   sleep(.75);
   serial_write_byte(204);
   serial_write_byte(0x0D);
   sleep(3.75);
   serial_write_byte(205);
   serial_write_byte(0x0D);
   sleep(.5);
   while(0)
     {
      for(i=0; i<255; i++)
        {
         serial_write_byte(i);
         printf("W: %x = %d\n", i, i);
         sleep(.75);
       }
    }

}

void test90()
{
   int EncoderL = 0, EncoderR = 0;
   long time;
   disable_encoder(EncoderLeft);
   disable_encoder(EncoderRight);
   enable_encoder(EncoderLeft);
   enable_encoder(EncoderRight);
   reset_encoder(EncoderLeft);
   reset_encoder(EncoderRight);
   motor(MotorLeft, -100);
   motor(MotorRight, 100);
   time = mseconds();
```

```
   do
     {
       EncoderL = read_encoder(EncoderLeft);
       EncoderR = read_encoder(EncoderRight);
   }while((long)(EncoderL + EncoderR) < ROTATION / 2L);
   ao();
   time = mseconds() - time;
   printf("Left: %d\nRight: %d\nAverage: %d\n", EncoderL, EncoderR, (EncoderL + EncoderR)
/ 2);
   printf("Time (ms): %d", time);
   waitForButton(BUTTON_A);
   disable_encoder(EncoderLeft);
   disable_encoder(EncoderRight);

}

void rotation()
{
   int EncoderL = 0, EncoderR = 0;
   disable_encoder(EncoderLeft);
   disable_encoder(EncoderRight);
   enable_encoder(EncoderLeft);
   enable_encoder(EncoderRight);
   printf("Spinning Left\n");
   motor(MotorLeft, -100);
   motor(MotorRight, 100);
   while(analog(2) < 128);
   reset_encoder(EncoderLeft);
   reset_encoder(EncoderRight);
   printf("Encoders Reset\n");
   while(analog(2) > 128);
   while(analog(2) < 128)
     {
       if (read_encoder(EncoderLeft) > 30000 || read_encoder(EncoderRight) > 30000)
         {
           printf("%d : %d", read_encoder(EncoderLeft), read_encoder(EncoderRight));
           reset_encoder(EncoderLeft);
           reset_encoder(EncoderRight);
       }
   }
   EncoderL = read_encoder(EncoderLeft);
   EncoderR = read_encoder(EncoderRight);
   ao();
   printf("Left: %d\nRight: %d\nAverage: %d\n", EncoderL, EncoderR, (EncoderL + EncoderR)
/ 2);
   waitForButton(BUTTON_A);
```

```
    disable_encoder(EncoderLeft);
    disable_encoder(EncoderRight);
}

void pickUpDistance()
{
    int set = 0;
    printf("Pick Up Distance\nPress \"A\" to lower arm\n");
    waitForButton(BUTTON_A);
    ArmExtend;
    BaseMid;
    GripOpen;
    enable_servos();
    sleep(.75);
    do
      {
        GripOpen;
        ArmDown;
        printf("Place Ball in gripper and press \"A\"\n");
        waitForButton(BUTTON_A);
        GripClose;
        printf("Press \"A\" to accept\nPress \"B\" to try again\n");
        set = waitForAnyButton();
    }while(set != BUTTON_A);
    printf("Hold Ball in place and press \"A\"\n");
    waitForButton(BUTTON_A);
    GripOpen;
    sleep(.75);
    ArmUp;
    printf("Let go of ball and press \"A\"\n");
    waitForButton(BUTTON_A);
    init_camera();
    track_update();
    pickUpSize = track_size(ORANGE, 0);
    printf("Size: %d", pickUpSize);
    waitForAnyButton();
    disable_servos();

}
```

Bump_avoidance.ic

```
void bump_avoidance()
{
   initialize();

   while(1)
     {
      changeSpeed(100, 100);
      while(!SensBump);
      if(SensBL & SensBR)
        {
         Turn(90L, dirRand);
       }
      else
        {
         if (SensBL)
           Turn(30L, dirC);
         else
           Turn(30L, dirCC);
       }

   }

}
```

Functions.ic

```
void pinouts()
{
   printf("******************\n");
   printf("*     Pinouts     *\n");
   printf("*Drop Sensors     *\n");
   printf("* Front      %x *\n", DropFront);
   printf("* Left       %x *\n", DropLeft);
   printf("* Right      %x *\n", DropRight);
   printf("*Bump Sensors     *\n");
   printf("* Left       %x *\n", BumpLeft);
   printf("* Right      %x *\n", BumpRight);
   printf("*Encoders         *\n");
   printf("* Left       %x *\n", EncoderLeft);
   printf("* Right      %x *\n", EncoderRight);
   printf("******************\n");
   waitForAnyButton();
}

void serial_write_char(char string[], int length)
{
   int i = 0;
   for(i=0; i < length; i++)
     {
      serial_write_byte(string[i]);
    }
}

int absJ(int val)
{
   if (val > 0)
     return val;
   return -val;
}

int max(int a, int b)
{
   if (a > b) return a;
   return b;
}

int min(int a, int b)
{
   if (a < b) return a;
   return b;
}
```

```
void request_robot_number()
{
   write_s3(200, 0, 0);
}

void write_ball_count()
{
   write_s3(200 + robot_number, 1, local_ball_count);
}

void write_base_busy()
{
   write_s3(200 + robot_number, 2, 1);
}

void write_base_free()
{
   write_s3(200 + robot_number, 2, 0);
}

void write_ball_count_done()
{
   write_s3(200 + robot_number, 4, local_ball_count);
}

void write_request_help()
{
   write_s3(200 + robot_number, 5, 0);
}

void write_helped()
{
   write_s3(200 + robot_number, 5, 1);
}


void write_s3(int p1, int p2, int p3)
{
   serial_write_byte(p1);
   serial_write_byte(p2);
   serial_write_byte(p3);
}

void serial_thread()
{
```

```
int part1 = 0, part2, part3;
printf("Serial Thread Started\n");
serial_set_mode(BAUD_RATE);
while(1)
 {
   do
     {
      while(serial_buffer_count() <= 0);
      part1 = serial_read_byte();
   }
   while(part1 < 200);//Main function
   while(serial_buffer_count() <= 0);
   part2 = serial_read_byte();
   while(serial_buffer_count() <= 0);
   part3 = serial_read_byte();
   if (part2 == 0)//Used for intialize robot number
     {
      if (part3 == 0)
        {
         write_s3(200, 0, robot_number + 1);
         printf("Send Robot #%d\n", robot_number + 1);
      }
      else
        robot_number = part3;
   }
   else if (part2 == 1) //Receiving ball update
     {
      away_ball_count = part3;
      global_ball_count = local_ball_count + away_ball_count;
   }
   else if (part2 == 2) //Receiving base busy update
     {
      base_busy = part3;
   }
   else if (part2 == 3) //Cooperative mode - outdated
     {
      if (coop < 4)
        ball_coordinates[coop++] = part3;
   }
   else if (part2 == 4) //Receiving ball update
     {
      away_ball_count_done = part3;
      global_ball_count_done = local_ball_count_done + away_ball_count_done;
   }
   else if (part2 == 5) //Cooperative Help
     {
```

```
        coop_help = 1;
      }

    }
}

void initialize()
{
    int i = 0, x, y;
    if (SensDrop)
      {
        printf("Place Robot on ground.\n");
        while(SensDrop)
          {
            beep();
            sleep(.1);
        }
    }
    _shut_down_pid = start_process(_shut_down_task());

    //Enable both encoders
    enable_encoder(EncoderLeft);
    enable_encoder(EncoderRight);
    printf("Starting in : 5");
    for(i = 5; i > 0;)
      {
        tone((float)i * 1000., .1);
        sleep(.9);
        display_get_xy(&x, &y);
        display_set_xy(x - 1, y);
        printf("%d", --i);
    }
    printf("\n");
}

void Turn(long Degrees, int Direction)
{
    int dir = 1;
    long Steps = ROTATION * Degrees / 180L;
    long Current = 0L;
    if (Direction > 1)
      Direction = random(2);
    if (Direction == dirCC)
      dir = -1;
    changeSpeed(turnSpeed * dir, -turnSpeed * dir);
```

```
   while(Current < Steps){Current = (long)read_encoder(EncoderLeft) +
(long)read_encoder(EncoderRight);}
}

void MoveSteps(long steps, int speed)
{
   long Current = 0L;
   changeSpeed(speed, speed);
   while(Current < steps){Current = (long)read_encoder(EncoderLeft) +
(long)read_encoder(EncoderRight);}
}

void changeSpeed(int MotorSpeedLeft, int MotorSpeedRight)
{
   track_position();
   motor(MotorLeft, MotorSpeedLeft);
   motor(MotorRight, MotorSpeedRight);
   motorSpeedLeft = MotorSpeedLeft;
   motorSpeedRight = MotorSpeedRight;
}

void track_position()
{
   reset_encoder(EncoderLeft);
   reset_encoder(EncoderRight);
}

void _shut_down_task() //Botbal Function from Kiss Inistitute
{
   int i,j;
   while(!SensDrop);
   hog_processor();
   ao();
   disable_servos(); // delete this line if you want your servos to freeze but remain powered at the
end
   printf("Game over");
   i= 0;
   for (j= 256; j <= 1024; j+=256) {
      hog_processor();
      while (i < j) {
         kill_process(_shut_down_pid+(++i));
         kill_process(_shut_down_pid-i);
      }
      ao();
      disable_servos(); // delete this line if you want your servos to freeze but remain powered at
the end
```

```
    }
    serial_set_mode(0);
    printf(".\n");
}

int scale(int i, int org, int new)
{
    return (i * new) / org;
}

int scaleSM(int i, int intScaleFactor) //Scale screen to motor
{
    return scale((SCREEN_WIDTH / 2 - absJ(i)) * intScaleFactor, SCREEN_WIDTH / 2,
MAX_SPEED);
}

void waitForButton(int button)
{
    while (check_button(button));
    while (!check_button(button));
    while(check_button(button));
}

int waitForAnyButton()
{
    int i;
    int val=0;
    while (any_button());
    while (!any_button());
    while(any_button())
      {
        val = 0;
        for(i = 0; i < 8; i++)
          {
            if (check_button(arrButton[i]))
              val += arrButton[i];
          }

    }
    return val;
}
```