Table of Contents

[**Development Environment** 2](#_Toc403542055)

[Hardware Setup 2](#_Toc403542056)

[Software Setup 2](#_Toc403542057)

[**Infrastructure** 3](#_Toc403542058)

[**Libraries** 3](#_Toc403542059)

[Libcreateoi 3](#_Toc403542060)

[LibIMU 4](#_Toc403542061)

[**Communications** 5](#_Toc403542062)

[Message Set 5](#_Toc403542063)

[iServer 6](#_Toc403542064)

[iClient 6](#_Toc403542065)

[Connecting to the Creates 6](#_Toc403542066)

[**Inner Loop Control** 7](#_Toc403542067)

[**Inih – Initialization Files** 7](#_Toc403542068)

[**Camera *(planned)*** 8](#_Toc403542069)

# Development Environment

Each Gumstix is running a version of embedded Linux called Yocto. The recommended development environment is a Linux machine with the necessary library dependencies installed. At a minimum, this includes the libraries ncurses, pthread, math, and opencv (for the camera). A windows machine can be used for testing so long as the compile environment is set up. At this time, no cross compiler has been set up so the development machines are only being used to test compilation and then the code is transferred to the Gumstix for final compilation.

## Hardware Setup



The Gumstix communicates with the TurtleCore board over /dev/ttyO0.

The IMU is accessed through /dev/ttyUSB0 with a baud rate of 38400.

The camera is still under development but will likely show up in /dev/video(X).

When connected to the laptop, the Create can be accessed through /dev/ttyUSB0 with a baud rate of 115200.

## Software Setup

An SVN repository has been set up for the base code here: <https://svn.ngst.northgrum.com/repos/AutonomyChallenge/trunk/>

It is recommended that each team build upon the functionality in this framework but coming up with an alternative is also allowed.

**Caution**

Since this is C code, be careful about the naming of globally defined variables. If they are of the same type, the Makefile may not give a warning for duplicate names and we have run into run time errors where the wrong variable was being modified in different places of the code.

# Infrastructure

The infrastructure has been set up in a multithreaded manor with a thread for each sensor, a thread for iServer to receive inter-robot communications, a thread for iClient to send status, and a main thread for the autonomy code execution.

main.c is where everything is initialized and the main while loop for program execution resides. At startup, all sensors are initialized, threads are created, and any data initialization is performed. The plan is to leave the robot in this configuration until a start key is pressed to begin the autonomy code execution. Then a quit key (likely ‘q’) will be used to nicely terminate execution, shutdown all the threads, and perform memory cleanup.

# Libraries

The Creates are equipped with two libraries, Libcreateoi, and LibIMU. Libcreateoi is an open source library (with a few customizations) used to communicate with the on board create sensors over the serial connection. LibIMU is the library written to process the data from the ArduIMU specifically set up for the Creates.

## Libcreateoi

This open source library can be found online at: <https://code.google.com/p/libcreateoi/>

This library is a C code wrapper for the Create Open Interface serial interface, defined here: <http://www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Open%20Interface_v2.pdf>

This library has been modified to work within our framework and execute at a fixed 50Hz rate. The samples from the original online repository have also been updated and expanded upon. This library gives access to all the Create onboard sensors and allows for wheel control. The following main functions are available:

|  |
| --- |
| /\*\*  \***Initialize the multithreaded Create serial interface**  \*\*/ |
| int startOI\_MT (const char\* serial, sem\_t\* sem\_input);  /\*\*  \***Functions to drive the Create**  \*\*/  /\*\*  \* Drives the Create with the given velocity (mm/s) and turning  \* radius (mm). The velocity ranges from -500 to 500mm/s, with  \* negative velocities making the Create drive backward. The radius  \* ranges from -2000 to 2000mm, with positive radii turning the  \* Create left and negative radii turning it right.  \*  \* A radius of -1 makes the Create turn in place clockwise and 1  \* makes it turn in place counter-clockwise. Also, a radius of 0  \* will make the Create drive straight.  \*\*/ |
| int drive (short vel, short rad);  /\*\*  \* Moves the Create at the specified velocity and turning radius until the  \* specified distance is reached, at which point it will stop. The interrupt  \* flag is used to either ignore collisions or to terminate on collisions, i.e.,  \* when the bumper sensor is tripped.  \*\*/ |
|  |
| int driveDistance (short vel, short rad, int dist, int interrupt);  /\*\*  \* Moves the Create at the specified velocity and turning radius until the  \* angle is reached, at which point it will stop.  \*  \*\*/ |
| int turn (short vel, short rad, int angle, int interrupt);  /\*\*  \***Functions to read Create sensor data**  \*\*/ |
| int readRawSensor (oi\_sensor packet, byte\* buffer, int size); |
| int readSensor (oi\_sensor packet); |
| int getDistance (); |
| int getAngle (); |
| int getVelocity (); |
| int getTurningRadius (); |
| int getOvercurrent (); |
| int getBumpsAndWheelDrops (); |
| int getCliffs (); |
| int\* getAllSensors (); |
| /\*\*  \***Function to shut down the Create interface**  \*\*/ |
| int stopOI\_MT (); |

More helper functions are available and can be found in the library header file.

## LibIMU

The Creates have ArduIMU v3 sensors with firmware that is specific to the Autonomy Challenge application. The IMU is connected on /dev/ttyUSB0 and LibIMU is used to decode the data. The IMUs have GPS disabled (since the majority of development will be indoors). The IMUs also have a 3-axis magnetometer that is currently disabled. The magnetometer provides a means to correct drift in the yaw orientation that occurs over time, but calibrating the magnetometers and testing other drift reduction options is currently a work in progress. With the magnetometers turned off, and without drift correction, the current IMU settings report accurate changes in the yaw orientation angle (but the magnitude may be off due to drift). This yaw orientation is currently accurate enough for initial testing, but operators will have to make note that the observed yaw drift has been upwards of 15 degrees over 5 minutes of operation. The IMU also provides accurate roll and pitch orientation, body angular rates, and accelerometer data. Since the vehicles operate only in a flat plane, the roll and pitch orientation angles are not used in the inner loop. Even though the IMU accelerometers are accurate, the current levels of noise in the sensor readings make integration of the data for velocity or position information difficult without filtering and the addition of another source of information (such as GPS).

The following main functions are available:

/\*\*

\***Initialize the IMU interface (on /dev/ttyUSB0)**

\*\*/

int startIMU\_MT (char\* serial);

/\*\*

\* **Initialize the IMU along with a file to log data in csv format**

\*\*/

int startIMU\_File (char\* serial, char\* file\_name);

/\*\*

\***Accessor functions to return the latest IMU data**

\*\*/

float getRoll();

float getPitch();

float getYaw();

float getGyroX();

float getGyroY();

float getGyroZ();

float getAccelX();

float getAccelY();

float getAccelZ();

/\*\*

\***Shutdown the IMU interface**

\*\*/

int stopIMU\_MT ();

# Communications

Each Create is equipped with a wireless adapter for communication on the TurtleCoreWiFi network. Every robot has been assigned a static IP address and a unique hostname. On startup, the Creates automatically connect to the network and once the main program is run, they begin broadcasting status messages on the multicast address 225.0.0.37.

## Message Set

The Creates are currently only sending and receiving a single status message:

struct **create\_status** {

int pos\_x; //in millimeters

int pos\_y; //in millimeters

int heading; //in degrees

int velocity; //in mm/s

int category; // i.e. Friend/Enemy/Unknown

char uid[32]; //robot unique hostname

}

## iServer

iServer is the set of functions to open a multicast socket to receive status messages from the other robots in the fleet. This server will need to be expanded upon if new messages are added to the message set because it is currently only set up to receive a single message type. Main functions include:

/\*\*

\***Start the multicast server to listen on 225.0.0.37**

\*\*/

void startServer();

/\*\*

\***Shut down the server thread and perform clean up**

\*\*/

void closeServer();

/\*\*

\***Return the latest status for the given robot, NULL if not found**

\*\*/

create\_status\* getStatus(char\* uid);

The server is storing the latest status from each robot in a map. If a status with an unknown robot ID is received, the message will be thrown out. If new robots are added to the system, make sure to add them to the map in iServer.h.

## iClient

iClient is the set of functions to open a multicast socket to send status messages. The client can be easily expanded to send any message type; however, it is currently only sending Create status. Main functions include:

/\*\*

\***Start the client to send status on a multicast socket on 225.0.0.37**

\*\*/

void startClient();

/\*\*

\***Send the latest robot status (usually at 2Hz rate)**

\*\*/

void sendStatus(create\_status status);

/\*\*

\***Shutdown the client socket**

\*\*/

void closeClient();

## Connecting to the Creates

To access the Create wirelessly, open a terminal on a development machine that is connected to TurtleCoreWiFi and use the robot’s name to ssh into the Gumstix:

ssh [root@ZoomBot.local](mailto:root@ZoomBot.local)

Each Create has a login of “root” and no password.

On a windows machine, Putty can be used to access each robot using it’s IP address.

If wireless communication isn’t working, use the mini-usb to USB connector to connect the TurtleCore board to the development machine. Then open a terminal to view the robot using “Screen”:

screen /dev/ttyUSB0 115200

/dev/ttyUSB0 is the location where the robot is connected to the development machine. 115200 is the baud rate for the connection. This can also be done on a windows machine using RealTerm but the appropriate drivers need to be installed.

# Inner Loop Control

The functions exist to control the Creates in two methods, turn robot while stationary, and turn while driving. Both methods use the previously defined “drive” function (see Libcreateoi) that executes a wheel speed (vel) and turn radius (rad).

For stationary turning, the following function is used to control the vehicle’s orientation:  
int headingControl(float yaw, int heading);

This function will align the current “yaw” orientation angle with an input “heading” command. Based on the yaw orientation error, “headingControl” calls the “drive” function to achieve the heading command.  
TODO: Since this function will be used while the vehicle is stationary, the name will be changed to “yawControl”, and the “heading” input will be renamed “yaw\_cmd.” This document will be updated when changes have been tested.

For turning while driving, the following functions can be used:  
void SpeedHeadingCommand(float \*WheelSpeedCmd, float \*TurnRadiusCmd, float SpeedCmd, float HeadingCmd\_deg, float dist\_PnPe, float Speed, float Heading\_deg, float sample\_rate, int cycle\_count);

void PositionCommand(float \*SpeedCmd, float \*HeadingCmd\_deg, float \*dist\_PnPe, float Pn\_cmd, float Pe\_cmd, float Pn, float Pe);

The “SpeedHeadingCommand” function will generate the required “WheelSpeedCmd” and “TurnRadiusCmd” to input to the “drive” function to command the speed and heading of the Create. This function operates on “HeadingCmd\_deg” and “SpeedCmd” commands from the user. The “PositionCommand” function can also be used to generate the “HeadingCmd\_deg” and “SpeedCmd” commands based on error between the current Create position, “Pn” & “Pe”, and some input command, “Pn\_cmd” & “Pe\_cmd”. A current example of this can be seen by running the driveIMU demo program located in Libcreateoi/samples and selecting either the Position command mode or the Waypoint command mode.  
TODO: Set up examples or control modes in which these functions are called from the main.c infrastructure, rather than just from the standalone driveIMU demo.

# Inih – Initialization Files

An open source library, Inih, is being used to process initialization files in the ini format. At startup, each robot is currently initialized with an initial status which provides location and robot class (i.e. friend/enemy). The initialization files can be expanded upon to load any data that is needed at startup. An example initialization file is shown below:

[Initial Pos]

pos\_x=2000;

pos\_y=0;

category=0; //FRIENDLY

uid = "HammerBot"

# Camera *(planned)*

The available camera is a Logitec C270 HD Webcam. Right now, the necessary drivers still need to be incorporated into the kernel so that openCV can communicate with the camera. The plan for the teams will be to use the openCV library to access the camera and then potentially use an open source library to process April tags that are found throughout the arena. All this is still being worked.