# Real Autobots and Decepticons

# Project Document

# Version 6.0

Prepared by

**Carter Chase**

**Trevor Sietz**

**Ben Smith**

**Jeff Ross**

Prepared for

# The Analogical Constructivism and Reasoning Lab

# Table of Contents

Introduction 4

Development Phases 4

Subphases 5

Project Development Summary 6

Preliminary Phase 6

*Project Proposal* 6

*Familiarity with Simulator* 6

Feasibility Phase 6

*Choose Robot Prototype* 6

*Research Arduino-Controller Communication* 7

*Research Simulator*-*Controller Communication* 7

*Develop Preliminary Robot Model* 8

Communication Phase 9

*Implement Arduino-Controller Communication* 9

*Implement Simulator*-*Controller Communication* 10

*Investigate Controller Architecture* 10

Movement Phase 11

*Assemble Arena* 12

*Investigate Arena Cameras* 12

*Develop Basic Robot Internals Library* 13

*Finalize Controller API* 13

*Move Robots from Controller* 13

AI Phase 14

*Enhance Robot Internals Library* 14

*Enhance Robot-Controller Communication* 14

*Research AI Planning Algorithms* 14

*Develop World Model* 14

*Develop AI Planning Algorithm* 14

Enhancement Phase 14

*Enhance Algorithm* 14

*Expand Formation Number* 14

Publicity Phase 14

*Team Poster at Research Expo* 14

*Department Poster* 14

*Demonstrations* 14

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Changes** | **Authors** | **Date** |
| 1 | Rough Draft | Carter | 09/27/2016 |
| 2 | Updated feasibility phase | Carter | 10/04/2016 |
| 3 | Updated Communication Phase | Carter | 10/25/2016 |
| 4 | Updated Communication Phase | Carter | 12/06/2016 |
| 5 | Updated Movement Phase | Carter | 12/07/2016 |
| 6 | Final Updates at end of Semester 1 | Carter | 12/08/2016 |

# Project Document

# Introduction

The goal of our project is to achieve the unaided assembly of modular robots into a predetermined shape from random starting points. In simple terms, an AI should be able to command a group of robots to arrange into a chosen shape (like a plus or a square).

This document outlines the team’s project progress for the self-reconfiguring robot research project conducted by senior students from the IPFW Department of Computer Science. The purpose of this document is to:

* Identify the phases of the project development
* Define the purpose of each phase
* Detail results of each phase
* Document any roadblocks and solutions

## Development Phases

The project consists of 7 nonconsecutive phases. These phases are detailed below and shown in Figure 1.

* **Preliminary Phase**: Become familiar with the problem and gather requirements.
* **Feasibility Phase**: Determine what is and is not possible.
* **Communication Phase**: Implement the program layer for communicating with the robots.
* **Movement Phase**: Implement the program layer for moving the robots.
* **AI Phase**: Implement the AI.
* **Enhancement Phase**: Enhance the features implemented in the previous phases.
* **Publicity Phase**: Publicize results.

Figure Phases of development

## Subphases

Each phase consists of several subphases whose timeline is documented in Figure 2 below.

Figure Subphases of development

# Project Development Summary

Each of the 7 phases has a specific purpose, with an ultimate result of achieving the project goal. This section gives detailed descriptions of each phase and documents any major discoveries from them.

## Preliminary Phase

The goal of the preliminary phase was to determine the basic requirements of the project and to familiarize the team with the available technology. It consisted of two subphases that are detailed below.

### *Project Proposal*

Completed by: Carter, Ben, Trevor, Jeff

The project proposal was a document required by the computer science department. The proposal that was submitted was an extension of the original proposal created by Dr. Licato. The proposal was extended by gathering new requirements and narrowing the project scope. A project goal was developed and agreed upon by the team. The phase finished with a presentation of the completed proposal to a group of faculty and fellow CS students.

### *Familiarity with Simulator*

Completed by: Ben, Jeff

Access was gained to a powerful 3D simulator called Webots. This simulator could be used to interact with virtual models of robots. Time was spent familiarizing the team with the software’s capabilities. The phase was completed by successfully importing the team’s robot model and interacting with it.

## Feasibility Phase

The goal of the feasibility phase was to research and test what is and isn’t possible. The team emerged from this phase with functional prototypes and a reasonable understanding about approaches to future phases. This phase was divided into four subphases which are described below.

### *Choose Robot Prototype*

Completed by: Carter, Ben, Trevor, Jeff

Before beginning any implementation, a robot type needed to be determined. By the end of the summer it was believed that the robot had already been finalized. The team was under the impression that the engineers were going to complete a prototype called the SMORES robot. Unfortunately, the engineers believed that the team had already agreed on their “block bots.” After several meetings with the engineering team, the miscommunication was recognized. A team meeting was held and a robot type was determined. The team proposed two robots, a “dream” robot and a “backup” robot.

* Dream: The engineers finish the SMORES robots.
* Backup: Roombas are purchased and configured by the team.

A SMORES prototype was promised by mid-October. The prototype would be evaluated and future prototypes built. Trust in the engineering team was not very strong, and a final prototype will be needed by January. It was decided to keep options open in case they didn’t come through, but development would continue assuming that the robots are completed on time.

### *Research Arduino-Controller Communication*

Completed by: Trevor

The first step in communicating with the Arduinos was to determine what type of connections it used. It was expected that the connection would be over Bluetooth using the Pybluez library to manage the Bluetooth connection. However, the Arduinos communicate using a subset of Bluetooth called BLE. From the program’s perspective, BLE behaves like a regular serial communication port connection. As a result, the project required a serial com library and not a Bluetooth library at all. Several serial communication ports libraries were investigated, however PySerial was the only one that was actively updated and supported. With that information, the project was limited to using PySerial to communicate with the Arduinos.

Originally, it was decided that multithreading would be the best way to communicate with the robots. This would allow for multiple comport to be read concurrently. There was one problem with multithreading serial communication ports, PySerial was not thread safe. According to its documentation, PySerial is currently under development to be thread safe and multithreading is actively discouraged. There was an alternative to multithreading: multiprocessing. Multiprocessing allows for multiple processes to be ran concurrently the only issue is the processes can share memory space making the program potentially unstable. The stability risk makes it advised to switch to switch to multithreading once PySerial becomes thread safe.

To allow communication between processes, it was determined that the python queue data structure contains the necessary concurrency protections. As a result, all communication between processes will be accomplished using the python queue. The final communication architecture can be seen in Figure 3.

### *Research Simulator*-*Controller Communication*

Completed by: Ben

Originally, it was determined that communication with the simulator should be *identical* to communication with a physical Arduino. Communication with the Arduino involved serial communication ports, therefore research was conducted into how to use serial communication ports with the simulator.

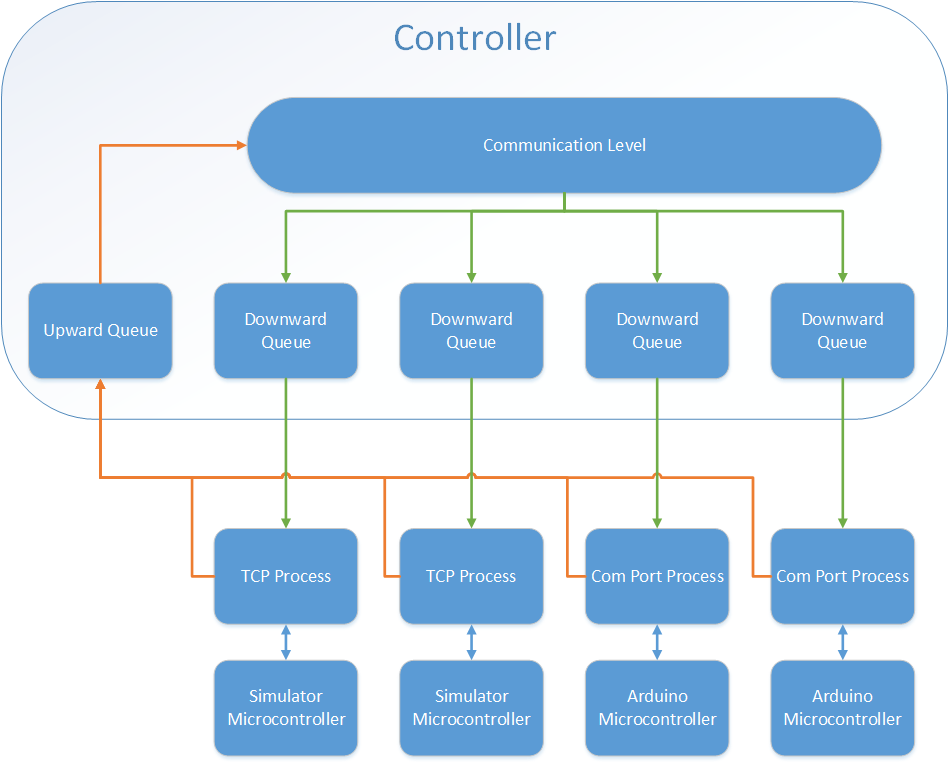
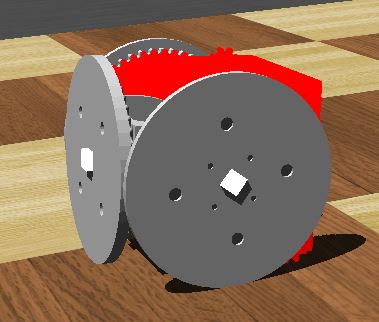
Continuous difficulties arose when using comports with the simulator because of some limitations of the simulator. In order to use serial communication ports in python it was necessary to import the PySerial module. Unfortunately, the simulator can’t import modules that are not already a part of the programming languages’ default libraries. The end result is a program the runs just fine outside of Webots, but crashes once loaded into a virtual robot.

Figure . Architecture for communication between the controller and the simulator/robots

After continued discussion it was agreed that another attempt should be made to use serial communication ports, this time in the C language. Even that proved futile, research revealed that recent updates in Windows 10 broke much of the support for serial communication ports. After this revelation, it was decided to abandon serial communication ports for the simulator altogether. A new approach was proposed using TCP ports in place of the serial communication ports. The architecture is shown in Figure 3.

### *Develop Preliminary Robot Model*

Completed By: Jeff

The engineering team provided a primitive model of the SMORES robot. Their model was accurate regarding the look of the robot and included some basic functionality such as wheel motors. Unfortunately, the model lacked several important devices, such as sensors and connectors, that would be needed for the simulation.

After replacing the missing components, an additional problem was apparent. The geometry of the model (which was also used for collision detection) dragged as the robot moved. This caused the robot to catch on the ground and veer off course. The simple solution was the creation of a simpler collision box.

Figure . Preliminary simulator SMORES model

The preliminary robot model can be seen in Figure 4.

## Communication Phase

The goal of the communication phase was to implement the communication level of the architecture produced in the previous phase. We hoped to emerge from this phase with functional code and a standardized API between the communication level and the movement level.

### *Implement Arduino-Controller Communication*

Completed By: Carter, Trevor

The main challenge in implementing this phase was determining timeout length. The Arduino is constantly accepting data from its comport. If the controller writes data too soon, the Arduino will never be able to finish the previous command. We realized that the Arduino programming was insufficient. It is designed to receive a numbered command (ex. 1 which is the command to move forward) and then perform the command for a set interval of time. This interval is hard coded and is different depending on the command given (ex. moving forward is set to two seconds, while turning is set to 1 second).

The team decided that the Arduino needed to be reprogramed to accept a time interval as well, so that the controller could specify the length of the operation. Unfortunately, due to time constraints, reprogramming the robots in the fall semester was not feasible. Therefore, controller was programmed with a hardcoded sleep interval. It was decided to reprogram the robot in a future task under the AI phase.

### *Implement Simulator*-*Controller Communication*

Completed By: Carter, Ben

Programming the simulator was no simple task. The simulator robots load separate controller programs that they use to control the virtual robots. These controller programs are extremely picky because of how the simulator robots interpret commands. The team tried several approaches to get the robots to respond to TCP commands, and all failed because the simulator could not understand how to interpret the controllers. Eventually, the team discovered that the simulator required that ALL motor commands must originate in a special event loop. This event loop keeps the robot online until there are no more commands for the robot to receive.

Unfortunately, running a TCP server, so that the robot can receive commands, actually blocks this event loop. There were two possible fixes to the problem. Option one involved spawning a separate process to host the TCP server and option two involved limiting the window where the TCP server receives requests. Investigation of option one revealed little fruit, however option two proved partially successful.

The flaw with option two has to do with command loss. Since the robot’s server only listens for commands when it is not actively doing something else, this resulted in early commands getting lost. The situation is as follows:

1. The robot receives a command from the controller
2. The robot processes the command and executes the orders
3. A new command is transmitted by the controller but the robot ignores it because it is already processing a command.
4. The controller waits for a response.
5. The robot finishes processing and listens for a new command (not realizing that one has already been sent and lost).
6. The connection times out.

This is not ideal at all. The connections fail between 10 and 20 commands. Ultimately the team will need to let the robot listen for commands at all times which will involve multiprocessing or the controller needs to resend commands after a period of time. Both approaches require extra research and will be enhancements in the AI phase.

### *Investigate Controller Architecture*

Completed By: Carter, Trevor

After lengthy discussion, the team decided to use a multiprocessing architecture for the controller. This meant that every level of the controller would be a separate process. The team decided to multiprocess the controller because of concerns over blocking the event loop. There was concern that the AI level in particular would spend too many processor cycles planning and would block the movement level and communication level from doing their actions. Additionally, the movement level would need to continuously process sensor data and update the world model which could also block the other levels.

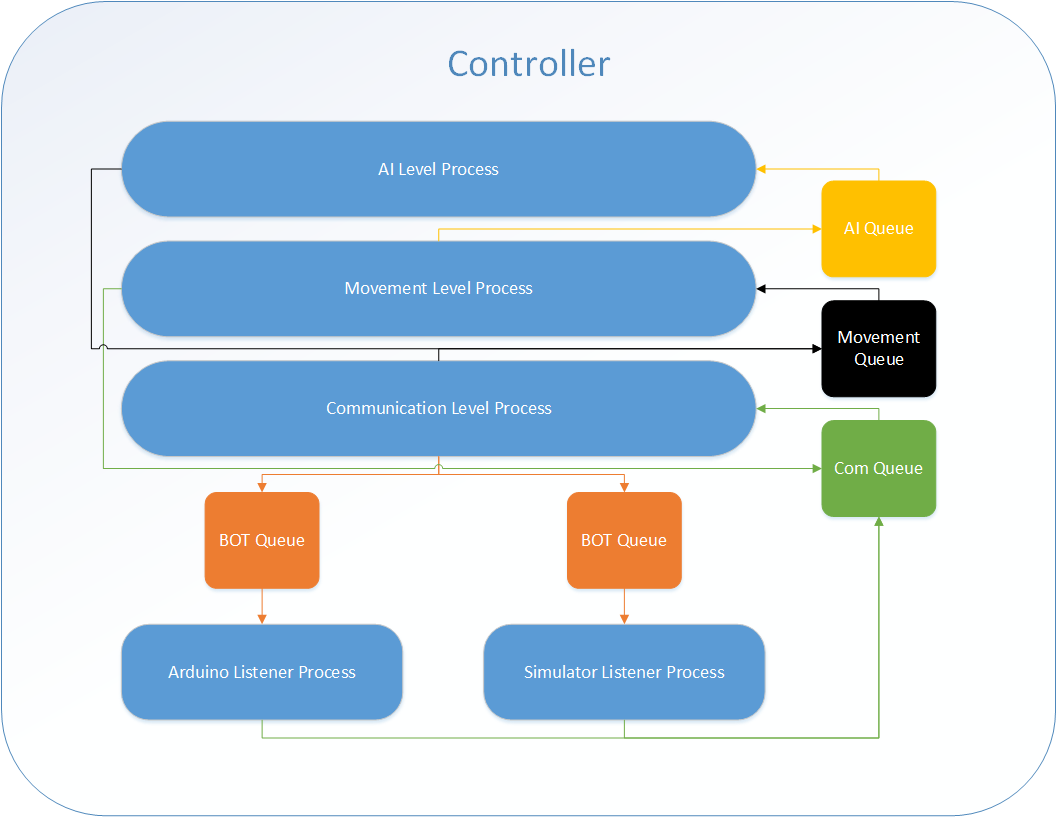
The different processes communicate between each other using Python Queues, which are memory safe. The messages put into the queue follow a universal format to ensure that each level can interpret a given message. The architecture of the queue communication can be seen in Figure 5. Not shown in Figure 5 is the “main” process. This process is responsible for interpreting information from the user and for managing the log messages form the different levels. It is also responsible for initiating an “elegant shutdown”.

Figure Multiprocessing architecture of the controller

## Movement Phase

The goal of the communication phase was to finalize top-down communication. The team wanted to be able to issue a command from the movement level and have it handed off to the communication level, relayed to the correct robot, and to have the robot correctly react to the command. The goal was to have everything in place for AI development in the AI phase.

### *Assemble Arena*

Completed By: Carter

A plywood arena was created with hardwood walls to contain the robots. The arena was rigged with 4 Pixiecams that were connected to two Bluno Nano microcontrollers. The completed arena can be seen in Figure 6.

Figure Assembled arena and cameras

### *Investigate Arena Cameras*

Completed By: Jeff

Little was known about the Pixiecams and how it reports data. Two weeks were spent investigating the capabilities of the camera as well as researching how the camera interacts with the Bluno Nano. The team ran into trouble getting the camera to interface with the Bluno Nano because there were no clear directions about which pins needed to be connected. After thorough research and consulting with the engineering team, the Pixiecam was finally interfaced with the Bluno Nano.

It was discovered that the Pixiecam does not report the actual pictures, instead it reports the location and size of objects that it has been trained to recognize. The primary metric it uses to recognize objects is color. The camera was to tested to see how well it detected emissive color, such as the colors produced when the robots activate their colored LEDs. Results of the testing showed that the cameras could recognize the LED color.

Finally the Bluno Nano was programmed to track the state changes of each camera and report the state to the controller when requested.

### *Develop Basic Robot Internals Library*

Completed By: Carter, Trevor, Jeff

The engineers provided a basic library for the robots. Time was spent understanding how the devices worked and how to issue them commands. Further investigation revealed that reprogramming the microcontrollers would be necessary. See the see the“Implement Arduino-Controller Communication” subphase for more context on this decision.

Time was also spent programming the virtual robots to follow the same API as the physical robots. The simulator robots will need to be updated as the physical robot’s API is updated.

### *Finalize Controller API*

Completed By: Carter, Trevor

Once the controller was broken into multiple processes, this subphase became even more important. The API will be used throughout the controller between every process. It was decided to use a dictionary data structure with a defined set of keys.

The keys are as follows.

{

'type': '', // log, error, sensor, command,

'origin': '', // the process where the message originated

'destination': '', // the process where the message is destined

'message: '', // the command/log for the destination to interpret.

'data': ''. // Additional data that the destination may need. (OPTIONAL)

}

The full Controller API can be seen at the following link.

<https://github.com/car-chase/amoebots/wiki/Controller-Interprocess-API>.

### *Move Robots from Controller*

Completed By: Carter, Trevor, Ben

The team met for a full day to combine the results of the previous subphases in an attempt to control the robots from the controller. The team was denied access to the robot due to miscommunication with the Engineering Department. In order to simulate the robot’s responses, a spare Bluno Nano was programmed to turn on lights in place of the motors. After reprogramming the microcontroller, attempts were made to interface it with the controller. First attempts revealed flaws in the original implementation of the communication level. After several revisions, the team was able to illicit a response from the microcontroller.

A second meeting was held with the robot and, after minor adjustments, it reacted to the same commands. Additionally, the simulator was tested and responded to the controller (see findings from “Implement Simulator-Controller Communication” for details on the limitations of the current implementation). Video of the results can be seen at the following link.

<https://youtu.be/B7y5TiiGJSY>

## AI Phase

The goal of this phase was to implement the AI algorithms that provide pathfinding and robot localization. It included enhancements to the robot APIs and Simulator.

### *Enhance Robot Internals Library*

Completed By: Carter, Jeff

We made minor changes to API to allow for magnitude rather than duration. The original API followed the following format <Command Speed Duration>, this was inadequate because speed is not a constant between robots. The new API follows the format <Command Magnitude> where magnitude is cm or degrees depending on the command. This requires us to calculate motor constants for each robot, but makes the overhead on the AI much smaller.

### *Enhance Robot-Controller Communication*

Completed By: Carter

Consisted of adding a ping to the controller to allow it to detect when a robot goes offline. Also enhancements to make the controller more configurable and robust.

### *Research AI Planning Algorithms*

### *Develop Localization Algorithm*

### *Develop AI Planning Algorithm*

## Enhancement Phase

### *Enhance Algorithm*

### *Expand Formation Number*

## Publicity Phase

### *Team Poster at Research Expo*

### *Department Poster*

### *Demonstrations*