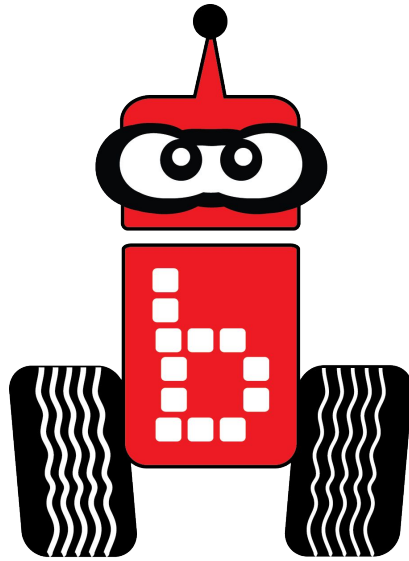
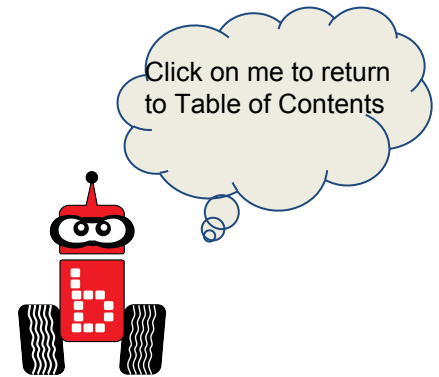




Using Servos

- **Key Concepts:**
 - Students will familiarize themselves with servo motors and how to control them.
- **Pacing:**
 - Over several class periods

Table of Contents



[Learning About Servo Motors \(Goals\)](#)

[Lesson: Learning About Servo Motors](#)

-Resources

[Activity 1: Intro to Servos](#)

[Servo Functions](#)

[Activity 2: `set\_servo\_position\(\)`](#)

[Activity 3: Wave the Arm](#)

[Activity 4: Hokey Pokey \(Dancing Robot\)](#)

[Activity 5: Touch the Can](#)

[Activity 6: Tag, You're Out](#)

[Assessment](#)

[What is a Claw?](#)

[Activity 7: Engineering a Claw](#)

[Activity 8: Using Two Servos](#)

[Activity 9: Go Fetch!](#)

[Activity 10: Recycle the Can](#)

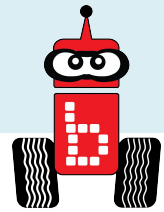
[Assessments](#)

[Back to the Drawing Board](#)

[Assessment Rubrics](#)

[Rubrics](#)

Learning About Servo Motors



Goals

- To distinguish between motors and servo motors
- To help students understand how to use Servo Motors with their robots
 - Enable, disable, set position and get position functions

Preparation

- Have KISS IDE up and running
- Have a robot ready to go
- Have a servo motor

Activities:

- [Learning About Servos](#)
- [Activity 1: Intro to Servos](#)
- [Activity 2: set_servo_position\(\)](#)
- [Activity 3: Wave the Arm](#)
- [Activity 4: Hokey Pokey \(Dancing Robot\)](#)
- [Activity 5: Touch the Can](#)
- [Activity 6: Tag, You're Out](#)
- [Activity 7: Engineering a Claw](#)
- [Activity 8: Using Two Servos](#)
- [Activity 9: Go Fetch!](#)
- [Activity 10: Recycle the Can](#)

Standards

Goal:

- Students will familiarize themselves with servo motors and how to control them.
- Students will familiarize themselves with the functions `msleep()` and `motor()`
- Students will understand how to move their robots in the following manner: forwards, backwards, straight, circles, right and left turns

Standards:

Common Core State Standards Math Practices

CCSSMP1: Make sense of problems and persevere in solving them

CCSSMP2: Reason abstractly and quantitatively

CCSSMP4: Model with mathematics

CCSSMP6: Attend to precision

CCSSMP8: Look for and express regularity in repeated reasoning

Next Generation Science and Engineering Practice

1: Asking questions and defining problems

2: Developing and using models

3: Planning and carrying out investigations

4: Analyzing and interpreting data

5: Using mathematics and computational thinking

6: Constructing explanations and designing solution

7: Engaging in argument from evidence obtaining, evaluating, and communicating information

Standards Continued

Standards Continued:

2016 ISTE Standards

Empowered Learner

1c: Students use technology to seek feedback that informs and improves their practice and to demonstrate their learning in a variety of ways.

1d: Students understand the fundamental concepts of technology operations, demonstrate the ability to choose, use and troubleshoot current technologies and are able to transfer their knowledge to explore emerging technologies.

Knowledge Constructor

3d: Students build knowledge by actively exploring real-world issues and problems, developing ideas and theories and pursuing answers and solutions.

Innovative Designer

4a: Students know and use a deliberate design process for generating ideas, testing theories, creating innovative artifacts or solving authentic problems.

4b: Students select and use digital tools to plan and manage a design process that considers design constraints and calculated risks.

4c: Students develop, test and refine prototypes as part of a cyclical design process.

4d: Students exhibit a tolerance for ambiguity, perseverance and the capacity to work with open-ended problems.

Computational Thinker

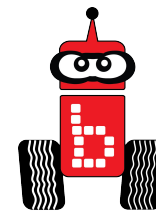
5a: Students formulate problem definitions suited for technology-assisted methods such as data analysis, abstract models and algorithmic thinking in exploring and finding solutions.

Prerequisite

Robot needs to have one servo with a 5 hole servo horn attached.

No claws!

Lesson: Learning About Servo Motors *



- A **servo motor** (or “**servo**” for short) is a motor that rotates to a specified **position between 0° and 180°**.
- Servos are great for raising an arm or closing a claw to grab something.
- Servo motors look very similar to non-servo motors, but there are differences...
 - A servo has **three wires** (orange, red, and brown) and a **black plastic plug**.
 - Servo motors only rotate 180° (not 360° like the non-servo motors)

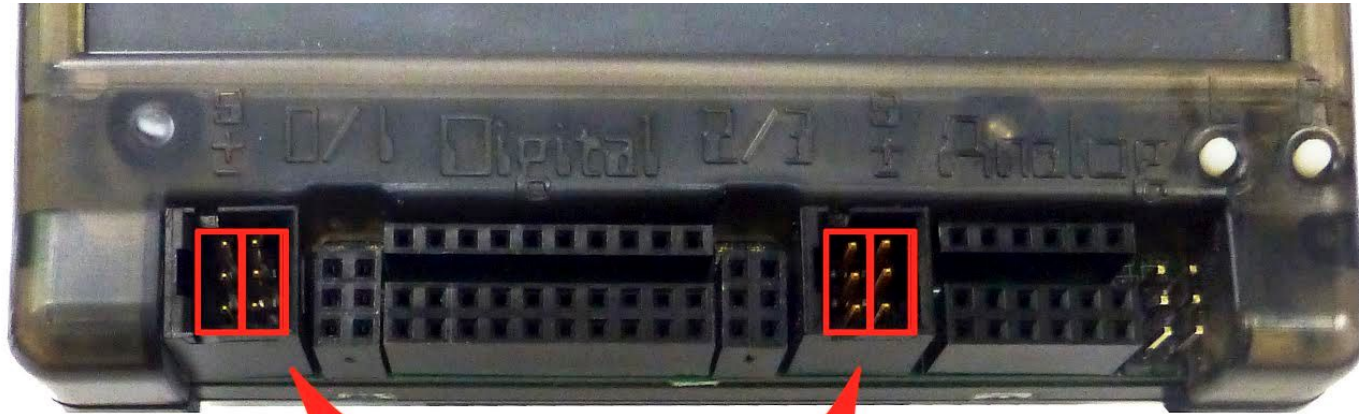
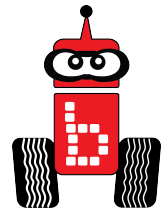
Large servo



*For more information about Servo Motors, refer to the *Sensor and Motor Manual* on the [KISS IDE Help](#).

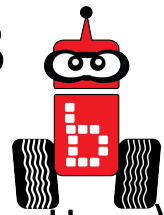
Learning About Servo Motors

KIPR Wallaby servo ports



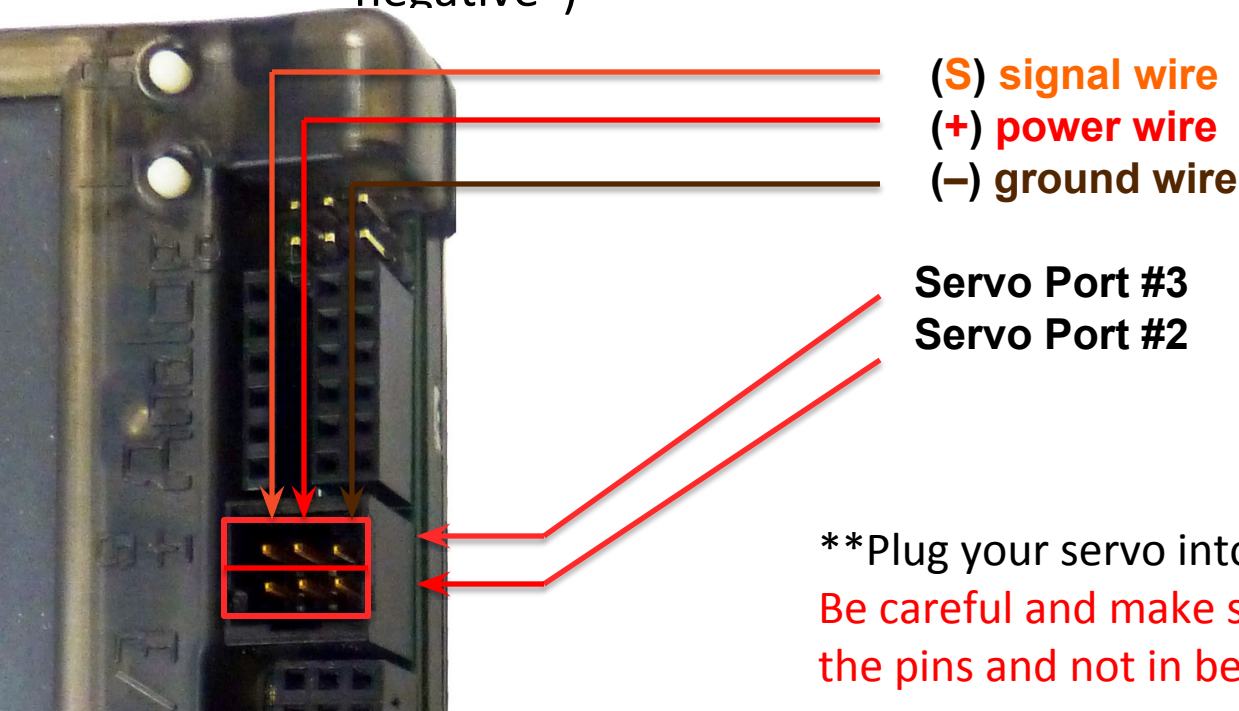
Servo Motor Ports 0, 1, 2, 3

Learning About Servo Motors



Plugging in Servos

- The KIPR Robotics Controller has 4 servo ports numbered **0** (bottom) & **1** (top) on the left, and **2** (bottom) & **3** (top) on the right.
- Notice that the case of the KIPR Robotics Controller is marked:
 - (**S**) for the **orange (signal)** wire, which regulates servo position
 - (**+**) for the **red (power)** wire
 - (**−**) for the **brown (ground)** wire (“the ground is down, down is negative”)



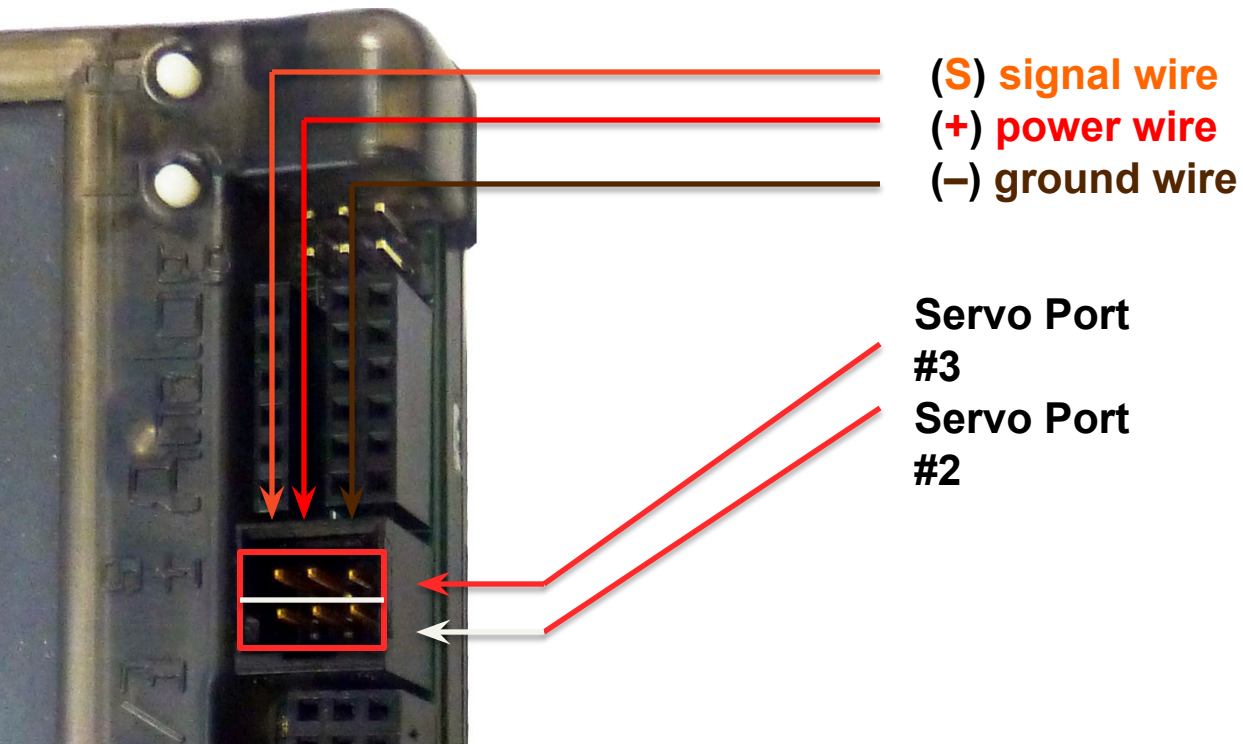
(**S**) **signal wire**
(**+**) **power wire**
(**−**) **ground wire**

Servo Port #3
Servo Port #2

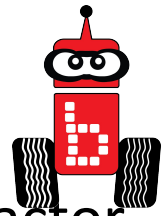
****Plug your servo into port 0**

Be careful and make sure to plug it into the pins and not in between the pins

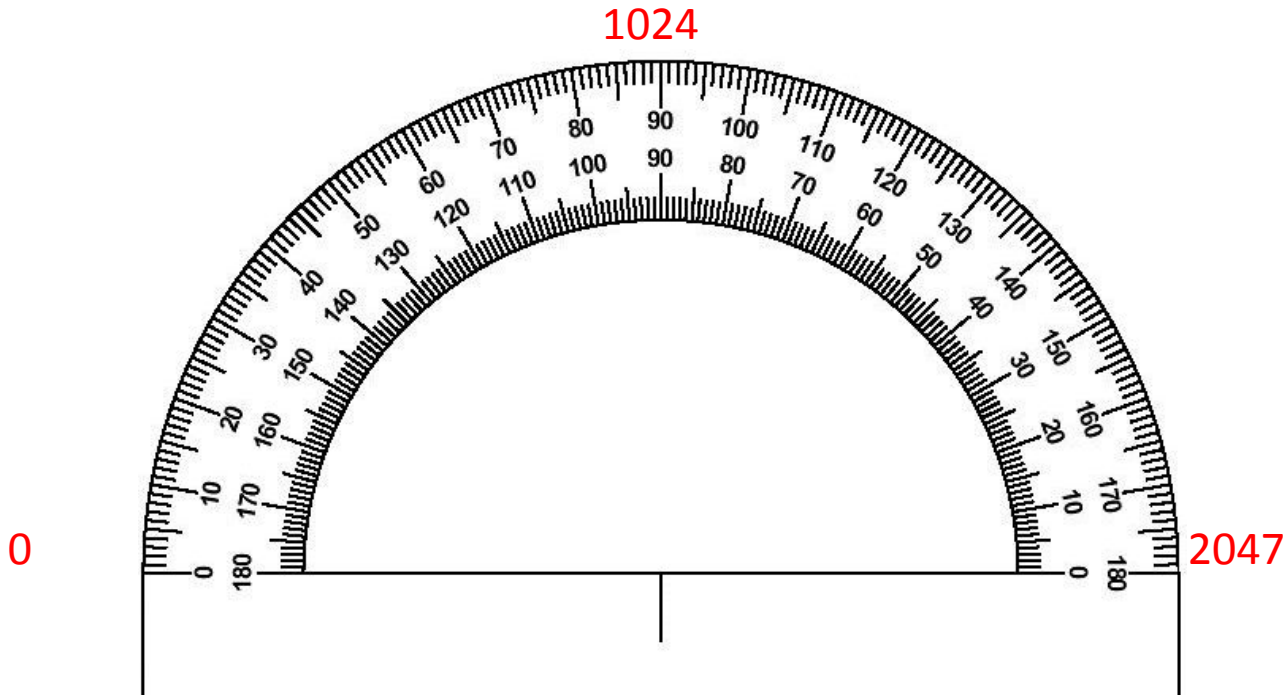
Plugging in servos



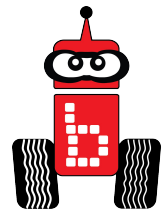
Learning About Servo Motors



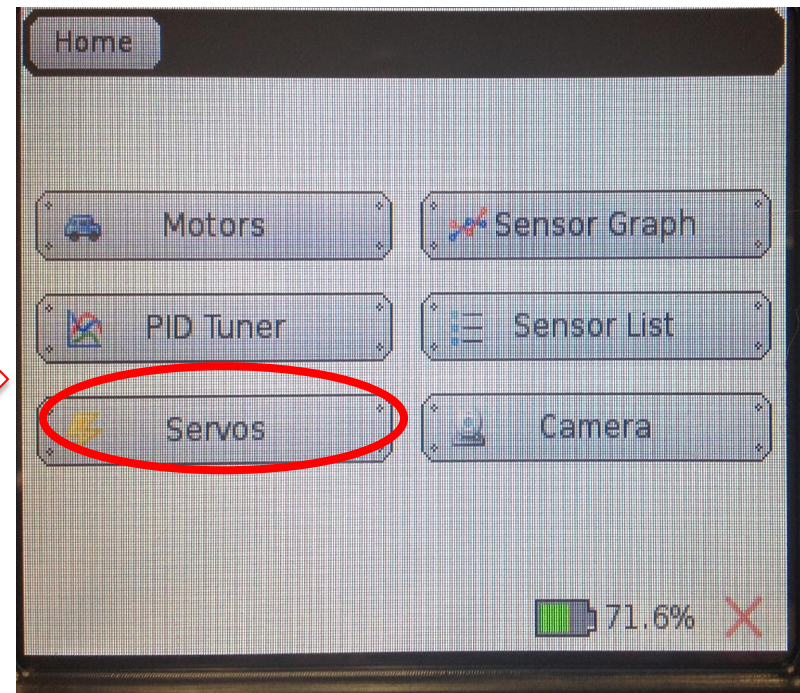
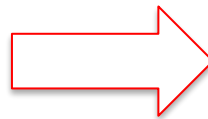
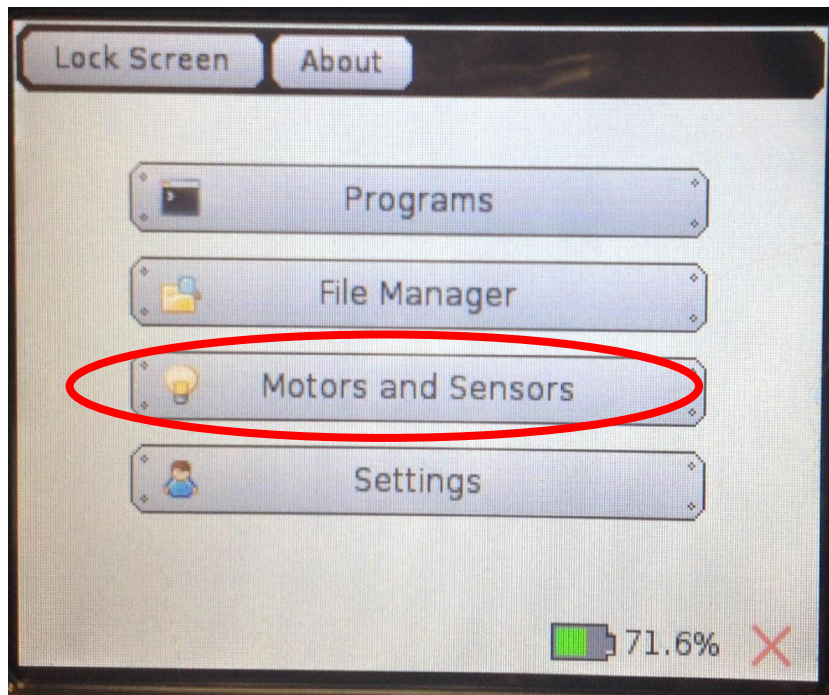
- If you think of servo motors range of motion like a protractor
 - The 180° is divided into **2048** positions (**0-2047**). Remember, we start counting with 0 and not 1
 - This allows for greater precision when setting a position (you have 2048 different settings you can choose)
- The default position is **1024** (centered)
- Your servo motor can only move $\frac{1}{2}$ a circle, (your wheel motors can move a full circle).



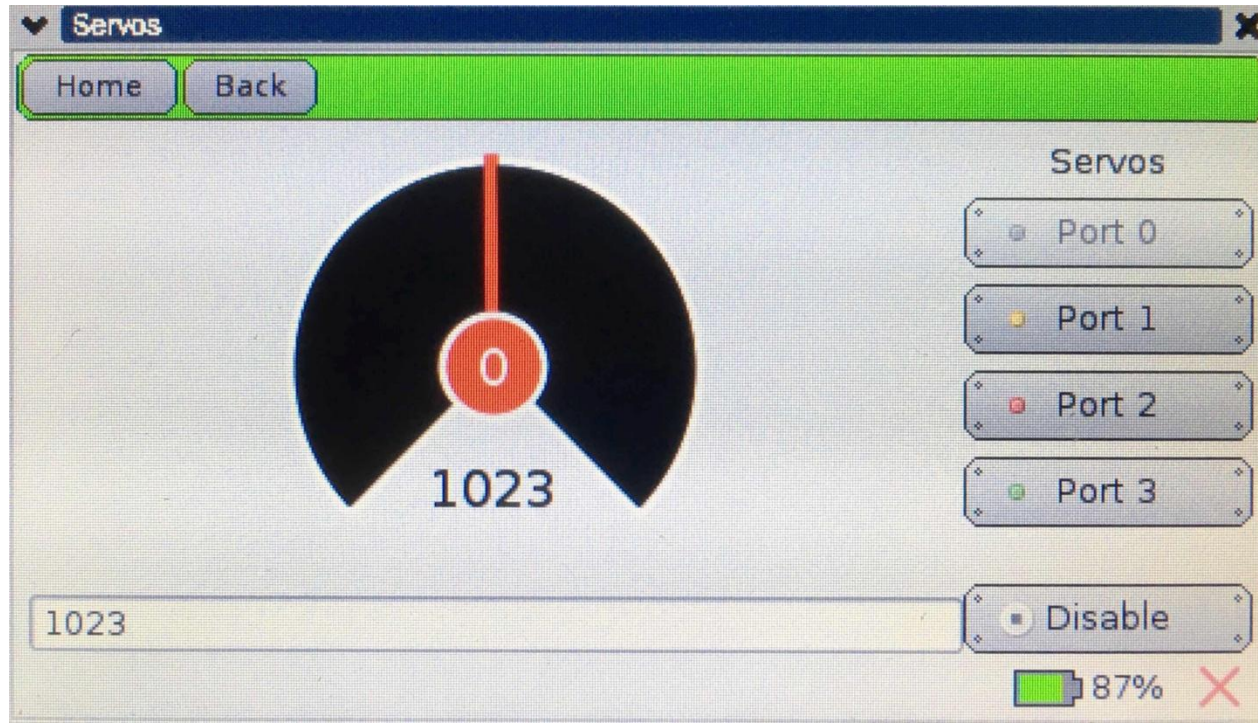
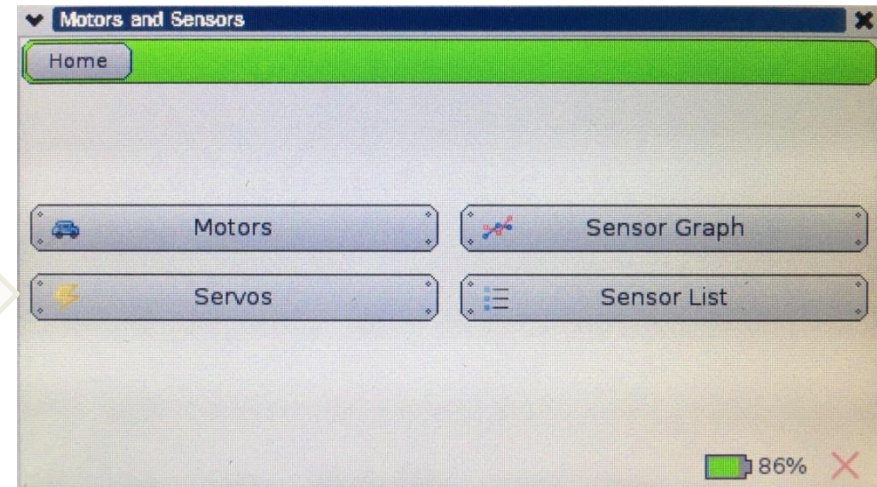
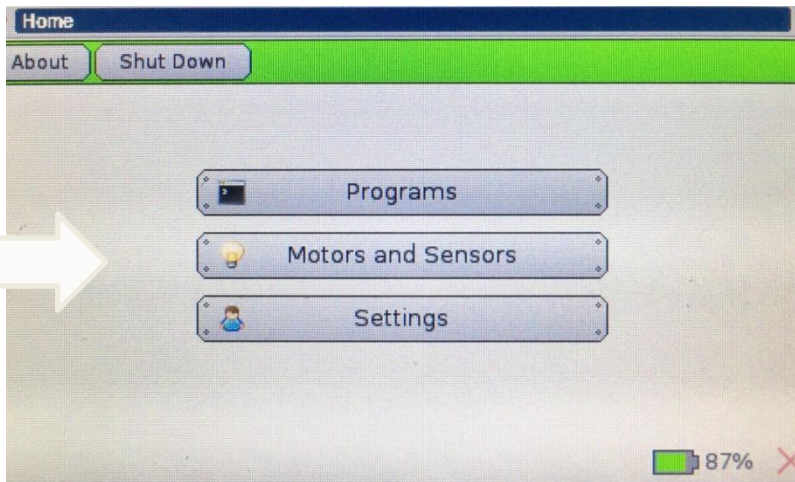
The Servo Screen



1. Make sure your Link is turned on
2. Plug a servo motor into Servo Port 0
3. Follow the guides to access the Servos Page on the wallaby



Use the Servo Widget

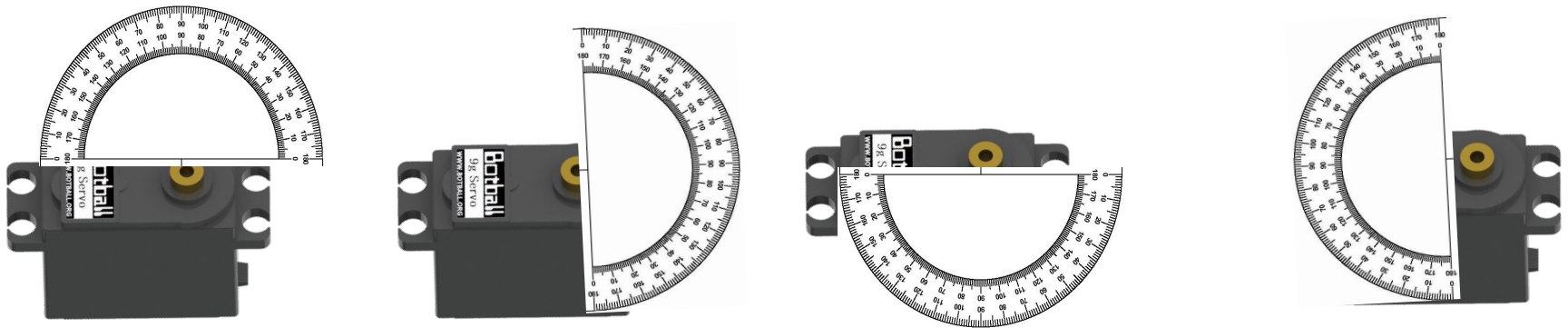


Centering Servo Horn

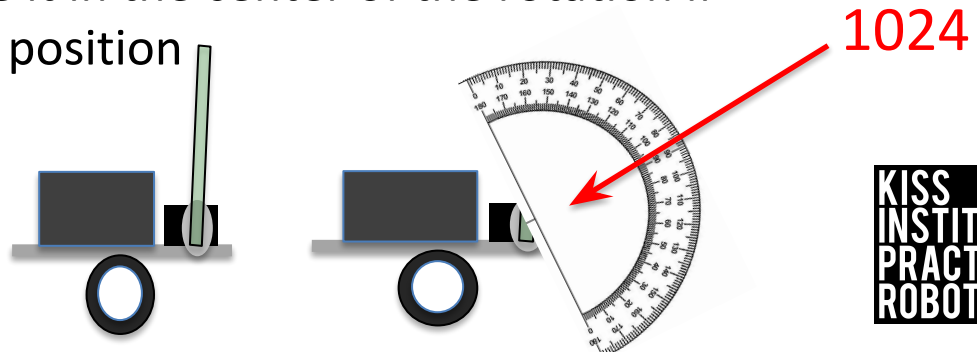
- The Servo motor only has a range of motion (rotates) ~ 180 degrees, but you cannot see by looking at the motor where this range of motion is located in relation to your robot.
- Using the Servo Widget, enable the servo on your robot. When you enable it, it will go to 1024. You can unscrew the servo horn on your arm or claw and place it in the center of the rotation if it is not already in the correct position.

Centering Servo Horn

- The Servo motor only has a range of motion (rotates) ~ 180 degrees, but you cannot see by looking at the motor where this range of motion is located in relation to your robot

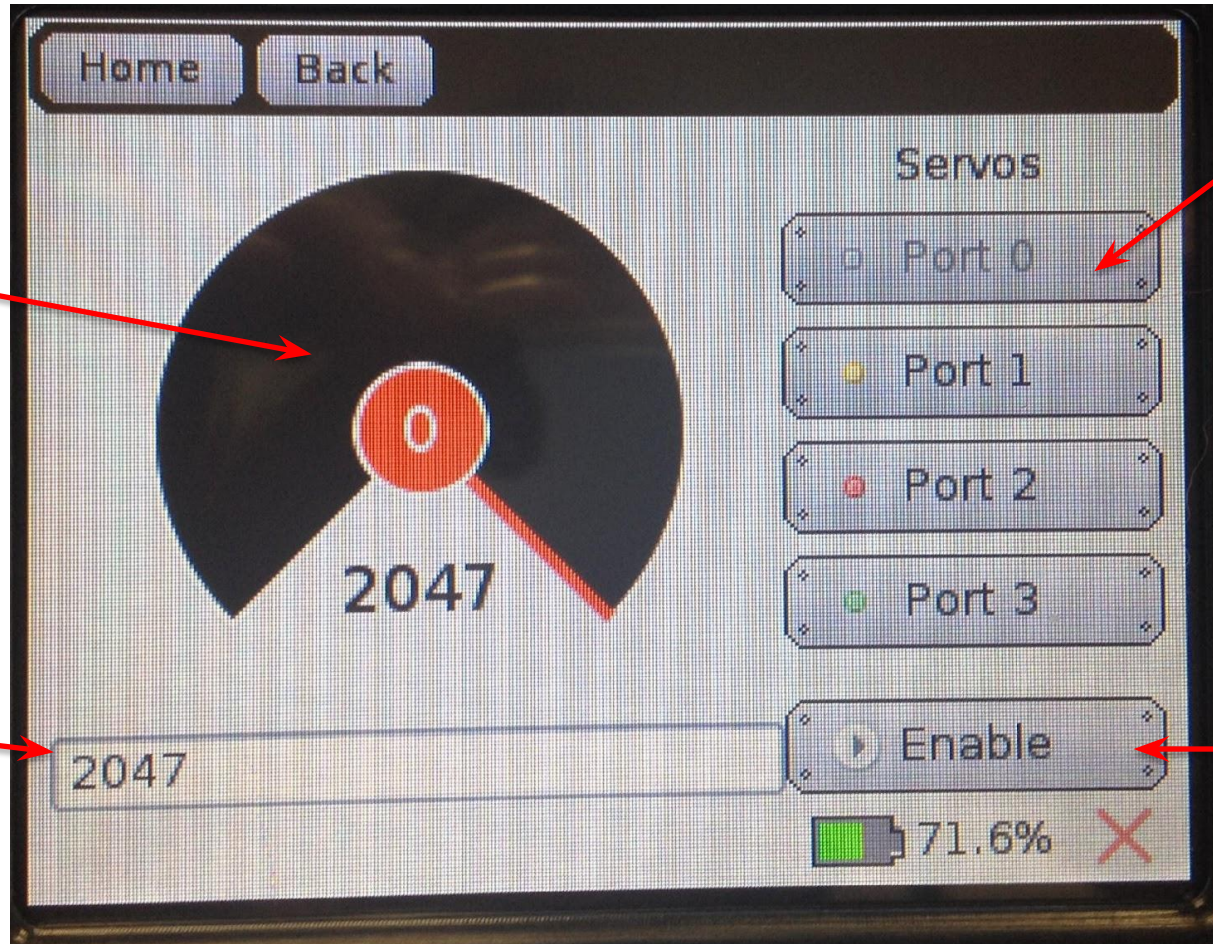


- Using the Servo Widget enable the servo on your robot. When you enable it, it will go to 1024. You can unscrew the servo horn on your arm or claw and place it in the center of the rotation if it is not already in the correct position



The Servo Screen (Cont.)

1. Use the Servo Page to test your Servo.

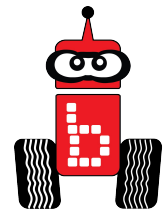


Select the
Servo Port

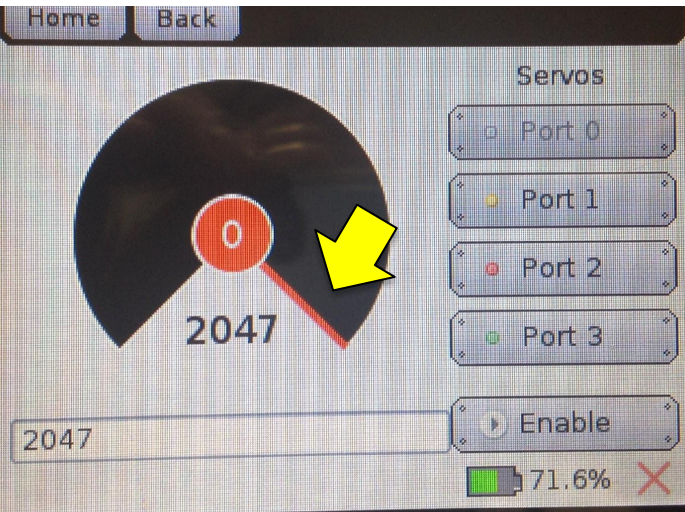
This is the
Servo
Position

Enables the
Servo

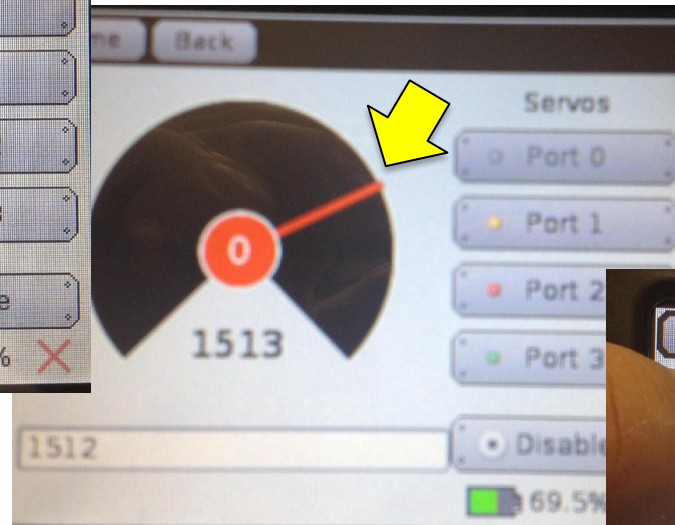
The Servo Screen (Cont.)



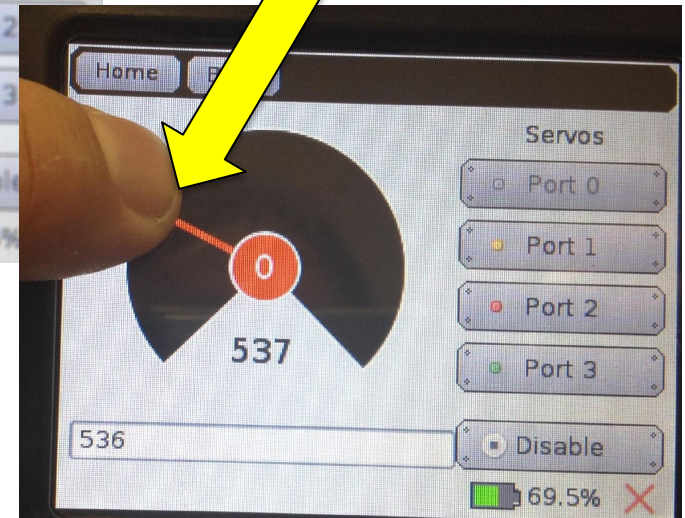
1. Use the Servo Page to test your Servo.



Servo
Maxed out
at 2047



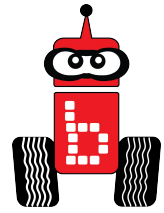
Servo at
1513



Servo at
537

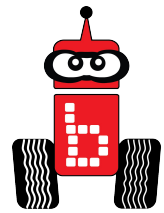
REMEMBER DO NOT KEEP PUSHING A SERVO BEYOND THE 0 OR 2047 POINTS- THIS CAN BURN SERVOS OUT!

Servo Functions



- To help save power, servo ports by default are **not** active until they are **enabled**.
 - Functions are provided for **enabling** or **disabling** all servo ports.
 - A function is also provided for **setting the position** of a servo.
-
- `enable_servos(); // Activate (turn on) all servo ports.`
 - `set_servo_position(2, 925); // Rotate servo on port #2 to position 925.`
 - `disable_servos(); // De-activate (turn off) all servo ports.`
-
- **Remember:** the range of positions is from 0-2047
 - The **default position** when servos are enabled is **1024 (centered)**, which means that all **servos will automatically move to this position** when `enable_servos` is called.
 - You can “**preset**” a servo position by calling `set_servo_position` **before** calling `enable_servos`. This will make the servo move to this position rather than center.

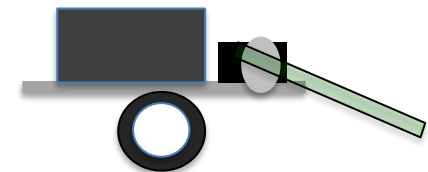
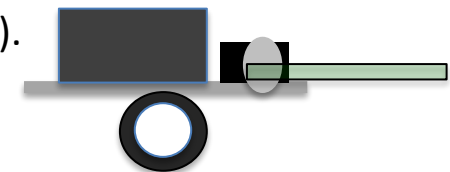
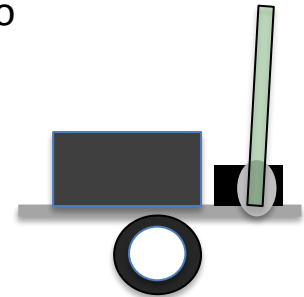
Intro to Servos



Materials: built robot with servo attached,
computer, notebook

Activity:

1. In your notebook, write down the port # the servo (arm) is plugged into or comment it within your program. `// arm = 0`
2. Set servo to up (vertical- pointing straight up) position. In your notebook, write down and label the up position or comment them within your program (see next slide) `// up = 1234`
3. Set servo position so that the arm is horizontal (level with the ground). In your notebook, write down and label the horizontal position or comment them within your program (see next slide).
`//Horizontal = 567`
4. Set servo position so that the arm is down (but not touching the surface). In your notebook, write down and label the down position or comment them within your program (see next slide).
`// down = 230`



Commenting your servo port and placement within your program

```
int main ()  
{  
  // arm = 0  
  // down = 230  
  // horizontal= 567  
  // up = 1234  
  printf("Sarah_Move_Servo");  
  return 0;  
}
```

Make your comments after the first curly bracket and before the print f

← Arm is plugged into servo port 0

← Arm down position is 230

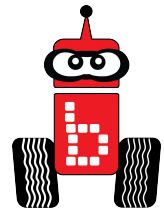
← Arm horizontal position is 567

← Arm up position is 1234

This can also be done in a notebook.

Activity 2

```
set_servo_position();
```



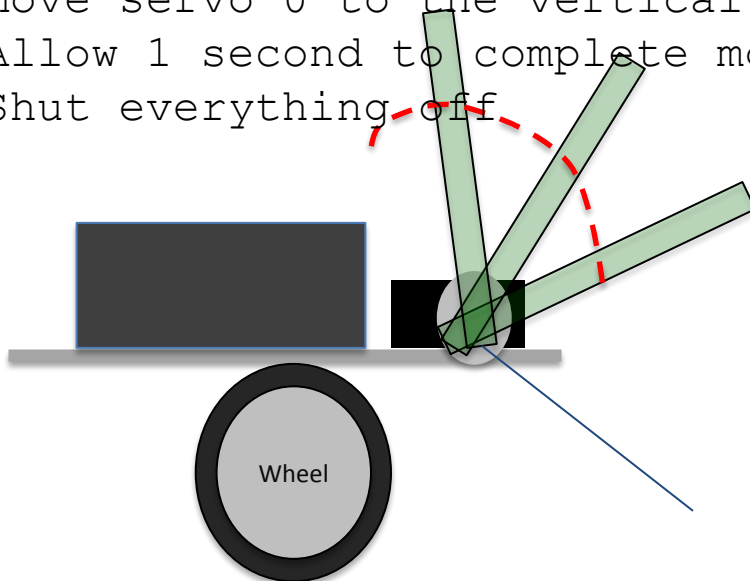
Create a new project in your folder: “Name” Servo

Replace the words “Hello World” in the printf function with “Name”, Servo

Write a program for your robot to:

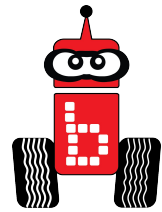
Pseudocode (Task Analysis)

1. Enable servos
2. Move servo 0 to the angled down position
3. Allow 1 second to complete moving to position
4. Move servo 0 to the horizontal position
5. Allow 1 second to complete moving to the position
6. Move servo 0 to the vertical (straight up) position
7. Allow 1 second to complete moving to position
8. Shut everything off

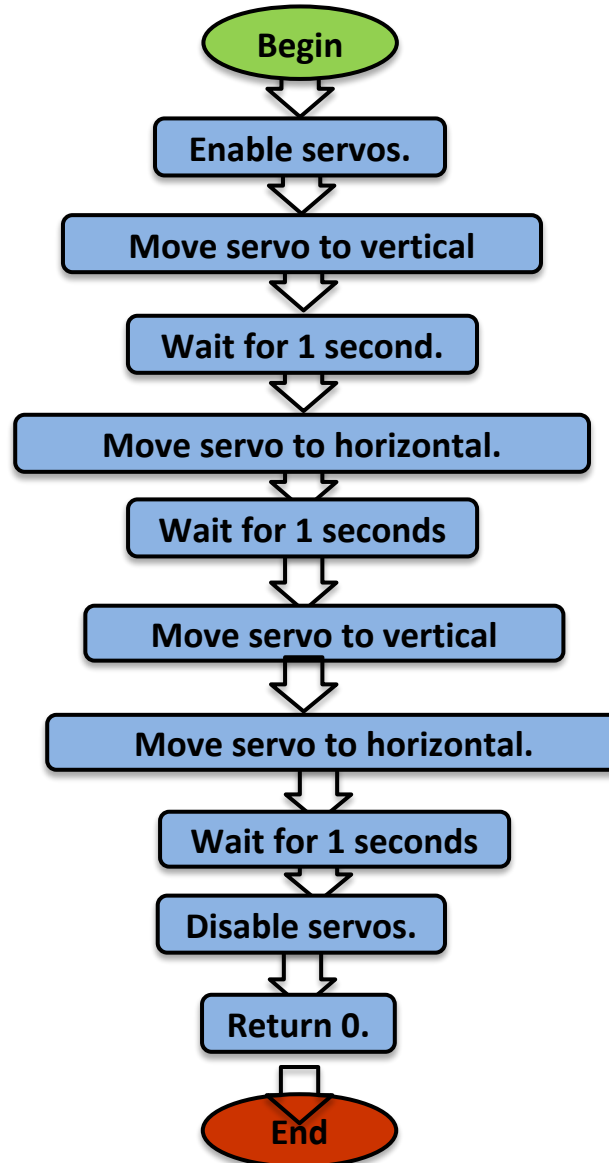


Activity 3

Wave the Servo Arm

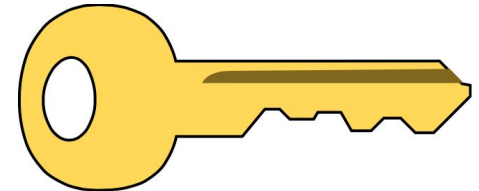


Flowchart



- Analysis:

You can use the JBC
Code Planning
Sheet to write your
pseudocode.



Click Key for Solutions

Possible Solution

```
#include <kipr/botball.h>

int main ()
{
    // arm = 0
    // up = 1234
    // down = 230
    // horizontal = 567
    printf("Sarah_Move_Servo\n");

    enable_servos(); //turn servos on
    set_servo_position (0, 230); // arm down
    msleep (500);

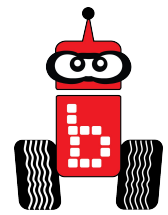
    set_servo_position (0, 567); // arm horizontal
    msleep (500);

    set_servo_position (0, 1234); // arm up
    msleep (500);

    disable_servos (); // turn servos off

    return 0;
}
```

Activity 4 Hokey Pokey (Dancing) Robot

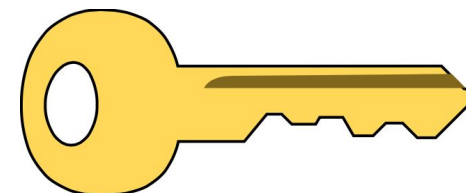


Create a new project in your folder: “Name” Hokey Pokey.
Replace the “Hello World” in the printf function with “Name”
Hokey Pokey.

Have the robots “dance” by moving their (one) servo and their
motors to the Hokey Pokey

You put your right hand in, `//Move servo to horizontal position`
You put your right hand out, `//Move servo to vertical position`
You put your right hand in, `//Move servo to horizontal position`
And you shake it all about, `//Move robot back and forth rapidly`

You do the hokey pokey,
and you turn yourself around, `//Turn robot in a circle`
That what it's all about.

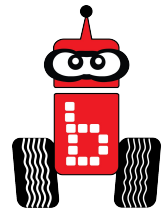


Click Key for Solutions

KISS
INSTITUTE
FOR
PRACTICAL
ROBOTICS

Assessment

Assessment 5: Dance Party



Setup: Use Surface-A. No game pieces required. Music required.

Goal: The robot must “dance” along with the music.

Limitations:

1. All robots must be autonomous (no remote controls, wireless communication, or touching the robot after starting a run).
2. The robot must start completely behind the vertical projection of the inside of the start line.
3. The students must provide their own music clip that plays loud enough for the judges to hear. Music clips can be played from a cell phone or the students can provide live music (singing).

Completion:

The robot must leave the starting box before completing the dance moves and must complete all of the following moves:

Must complete at least one 360 degree clockwise turn

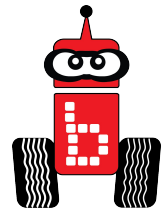
Must complete at least one 360 degree counter clockwise turn

Must move forward

Must move backward

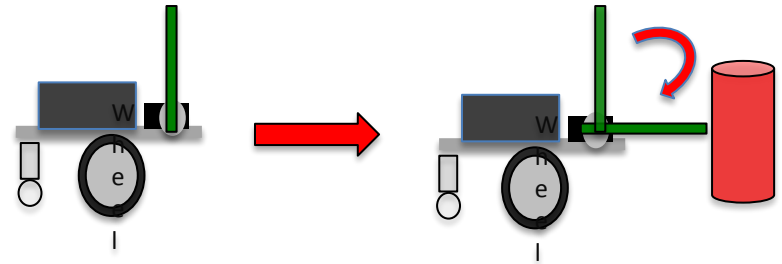
Must wave the servo (up and down at least once)

Activity 5: Touch the Can



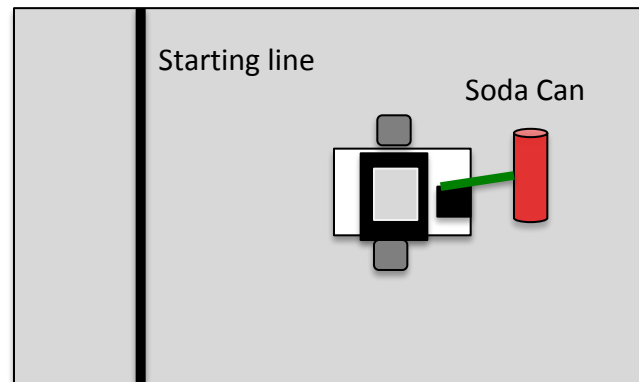
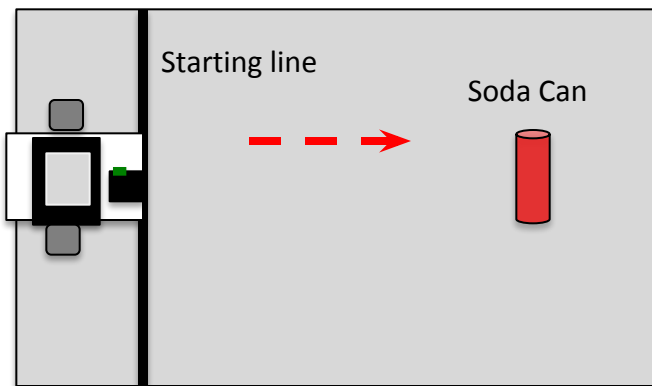
Create a new project in your folder: “Name” Touch the Can

1. Robots must start on or behind the starting mark and move to the object with the goal of touching the object WITH the LEGO attached to the servo in the shortest amount of time
2. The pointer must start in the vertical position and then move to the position required to touch the can

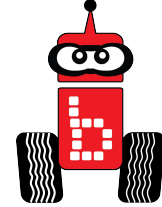


Extensions

1. Move the can to various distances
2. Make the object smaller and harder to navigate to
3. Math- have them measure the distance to the object and time the robot and then calculate rate/speed. $\text{Speed} = \text{Distance} / \text{Time}$.

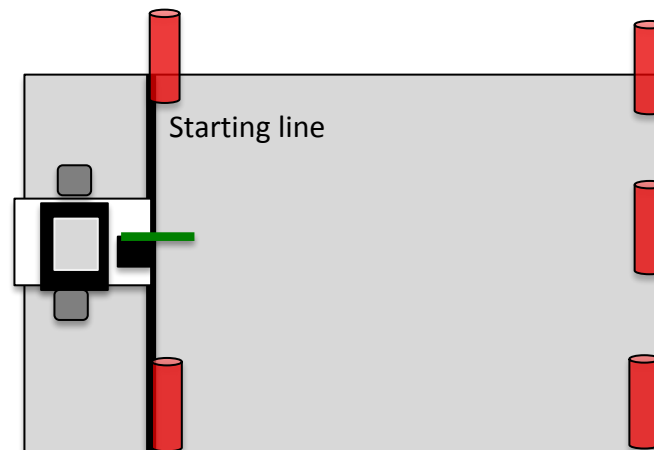
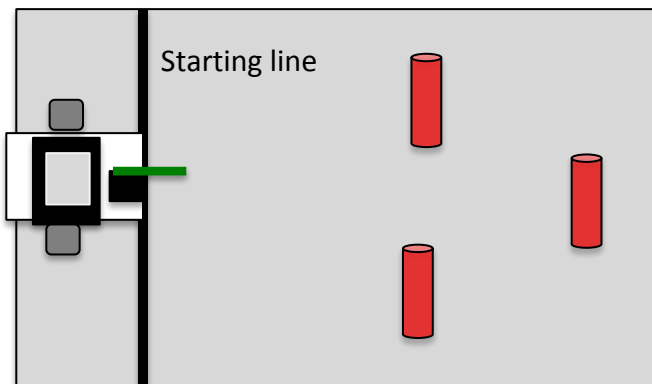


Activity 6: Tag, You're Out



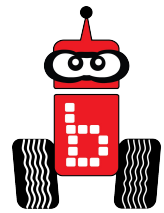
Create a new project in your folder: “Name” Tag You're Out

- “Tag” with your servo pointer the objects, which are then removed from the board
 - Must tag with the pointer only- if they touch it with any part of the robot other than the pointer it does not count
 - Pointer has to change position to tag (they can't drive around with the pointer out front all of the time)
 - Use the numbered circles on Surface A to place items.
- Score points for every item removed from the area



Assessment

Assessment 9: Add it Up



Setup: Use Surface-A.

Goal: Drive the robot to the numbered circles on the mat (doesn't have to be sequential), and then use a servo to touch the circles.

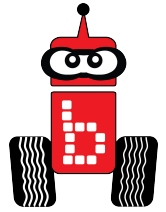
Limitations:

1. All robots must be autonomous (no remote controls, wireless communication, or touching the robot after starting a run).
2. The robot must start completely behind the vertical projection of the inside of the start line.
3. Robots must use a servo to lower an effector to touch the circle (it cannot be something that is always dragging or always touching the surface)
4. To count as touching a circle part of the robot must be lowered by a servo and touch either inside the red circle or on any part of the red circle line itself.
5. You can only touch one circle at a time.

Completion: When the robot drives around the can and returns behind the starting line.

Extra Optimization: Change the goal for the number of touch points accrued in a single run.

Introduction to a Variable:



Some reasons to use a variable:

1. You don't have to *remember* which port # is your arm and which is your claw—the computer remembers for you
2. You don't have to remember which servo position is up or down or open or closed
3. It makes your program easier to read and understand
4. Makes it easier to debug your program

Turning Your Comments into Code

```
#include <kipr/botball.h>
int main ()
{

// arm = 0
// up = 1234
// down = 230
// horizontal = 124

printf("Sarah_Move_Servo\n");

return 0;
}
```



Remove the
forward slashes
from your
comments, add
`int` and add the
semicolon

```
#include <kipr/botball.h>
int main ()
{

int arm = 0;
int up = 1234;
int down = 230;
int horizontal = 124;

printf("Sarah_Move_Servo\n");

return 0;
}
```

Possible Solution

```
int main ()
{

int arm = 0;
int up = 1234;
int down = 230;
int horizontal=124;

printf("Hello World\n");
enable_servos ();

set_servo_position (arm,down); // port 0, position 230
msleep (500);

set_servo_position (arm,horizontal); // port 0, position 124
msleep (500);

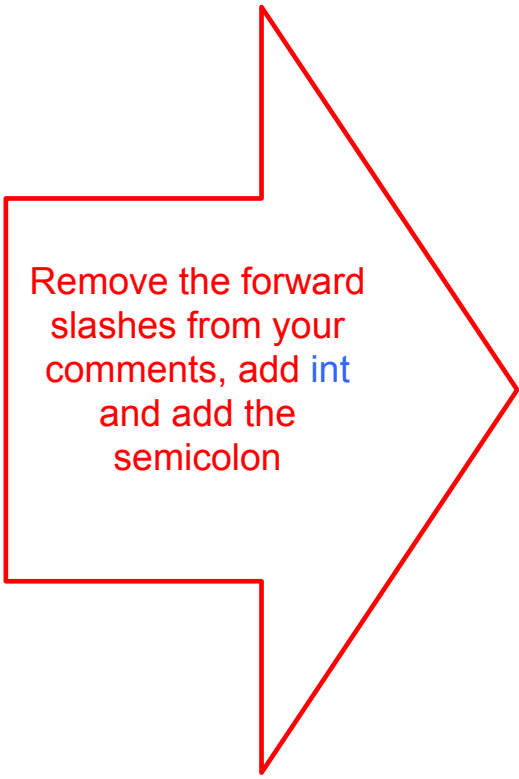
set_servo_position (arm,up); // port 0, position 1234
msleep (500);

disable_servos ();
return 0;
}
```

Drive Motors

1. A variable should go inside the int main block of code (i.e., inside the { }) immediately after the starting curly brace (i.e., {).

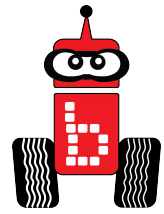
```
int main ()  
{  
  
    // left = 0  
    // right = 3  
  
    printf("Sarah_Drive\n");  
  
    motor (0,100);  
    motor (3, 100);  
    msleep (1000);  
  
    return 0;  
}
```



Remove the forward slashes from your comments, add int and add the semicolon

```
int main ()  
{  
  
    int left = 0;  
    int right = 3;  
  
    printf("Sarah_Drive\n");  
  
    motor (left,100);  
    motor (right,100);  
    msleep (1000);  
  
    return 0;  
}
```


What is a Claw?



Have you seen a claw working before?

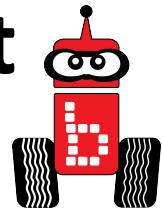
What job (function) does it complete?

Think about the claw.

How is it built (structure)?

- The claw is great for grabbing (function) objects.
- The claw can lift things up (function).
- The claw is not good at pushing things (function).
- What does it look like? (structure).

Moving Objects with Your Robot

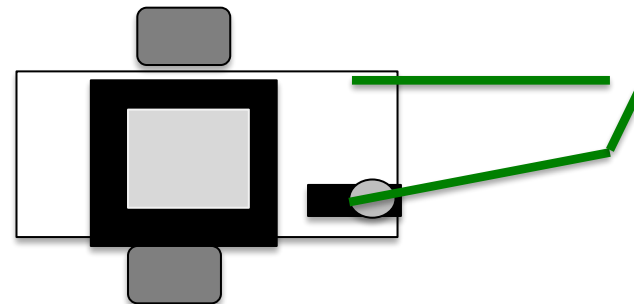
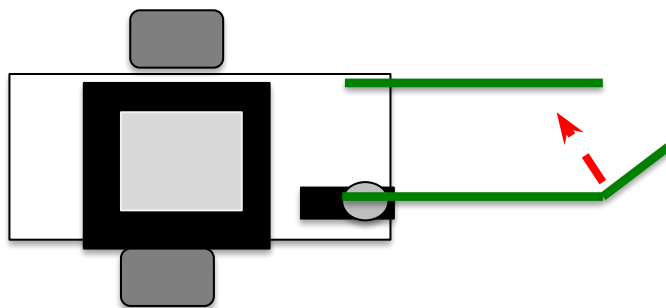


Claws/Grippers

Engineering*

A structure can be built onto the servo(arm) on your demo bot that can be closed and opened to grab an object

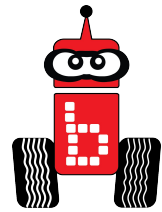
- You can build this out of LEGO and KMP
 - There are a lot of photos of claws and grabbers on YouTube, the Botball webpage and the Botball Educational Robotics Facebook page
- The easiest and first grabber to build has a static (unmovable) side and a side with a servo that closes
 - Write a function for opening and closing the servo
- You can use two servos, one to raise and lower the claw/gripper and one to open and close the claw/gripper



Activity 7 : Engineering a Claw

Go to the Robot Building Guide and follow the instructions to build a claw on your robot. Or use the Engineer Design Process to create your own.

Activity 8: Using Two Servos



Create a new project in your folder: “Name” Using two servos

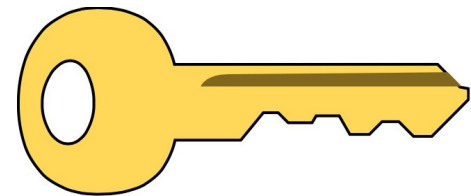
For this activity, you are going to grab the can with the claw without moving the robot.

Setup: The wallaby will be set with the claw horizontal and open. The can will be placed inside the claw. Comment the positions with the program (see next slide).

Pseudocode

1. Turn on servo
2. Claw horizontal
3. Claw open
4. Close claw
5. Arm up
6. Turn off servos

**** remember you must have wait time (msleep)



Click Key for Solutions

Commenting within your program

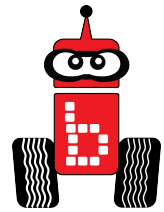
```
int main ()
{
// arm = 0  ← means arm is in port 0
// up = 1234 ← means up position is 1234
// down = 230 ← means down position is 230
// horizontal =1000 ← means position is 1000
// claw = 3 ← means claw is in port 3
// open =1246 ← means open claw position is 1246
// closed =34 ← means closed claw position is 34

printf("Hello World!\n");

return 0;
}
```

Commenting your positions
makes it easy to remember
the correct positions

Activity 9: Go Fetch!



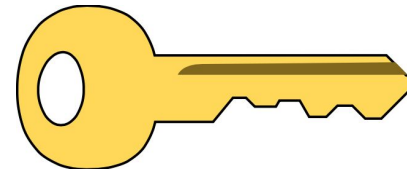
Create a project in your folder: “Name” Go Fetch

For this activity you will design a claw that will go out and pick up a can that is on circle #9 and return home. Hint: Great time to “Be the Robot”.

- Things to think about:
 - How many servos will you use?
 - Do you need to give the servos time?
 - When should you set the servo position?

[An extension for Using Variables](#)

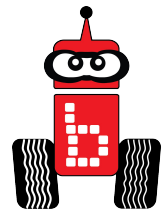
Click Key for Solutions



An Example of Pseudocode

1. enable
2. arm horizontal
3. open claw
4. forward can six
5. all off
6. close claw
7. up claw
8. back
9. disable

Activity 10: Recycle the Can



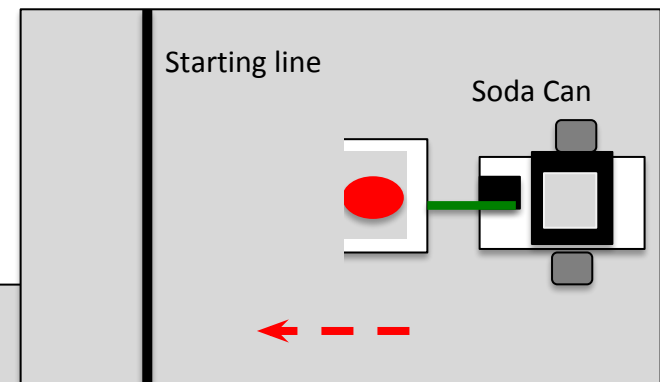
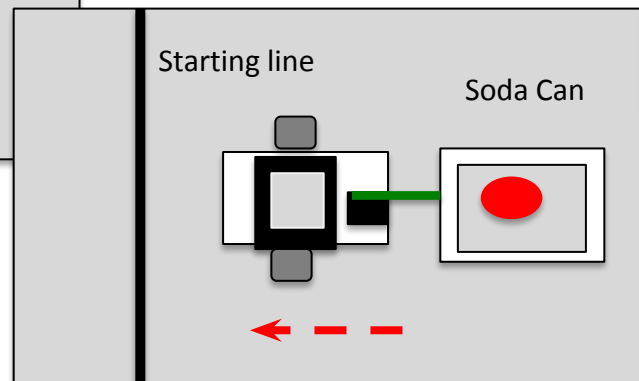
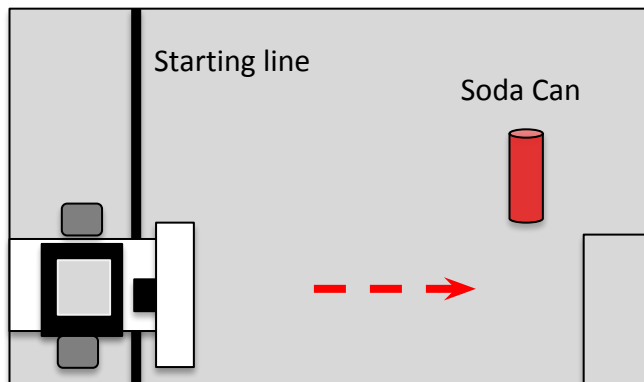
Create a new project in your folder: “Name” Recycle the Can

Robots must start on or behind the starting mark and move to the object with the goal of bringing the can back to the starting line.

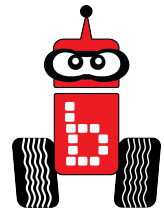
Make the arm/claw/grabber start in the upright position and then lower itself after starting or approaching the object.

Extensions

- Move the can to various distances
- Make the object smaller and harder to navigate to
- Math- have them measure the distance to the object and time the robot and then calculate rate/speed

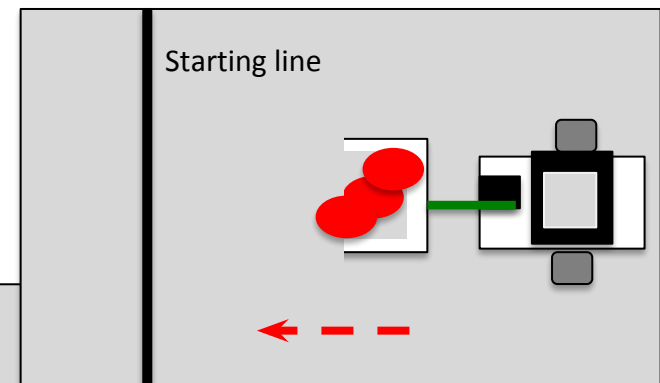
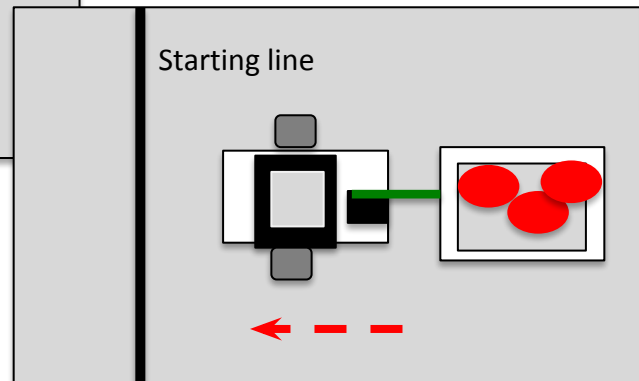
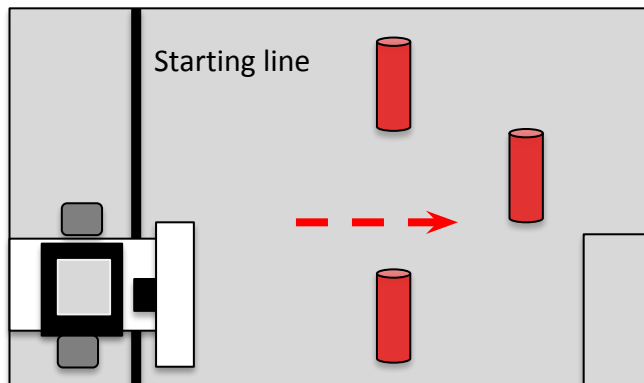


Recycle the Can(s)



Same as recycle the can only with more objects

- Place the items at known or set locations
 - This is because you are still “dead reckoning”, once we learn more logic and decision making, we can program smarter robots that will use sensors to locate and find the objects, which can then be tagged and removed.



Assessment

Assessment 21: Foot Tall

Setup: Use Surface-A.

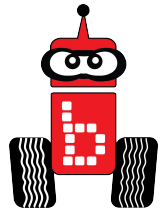
Goal: The robot will drive out to the can and lift the can so that the lowest part of the can is over 11 inches above the mat.

Limitations:

1. All robots must be autonomous (no remote controls, wireless communication, or touching the robot after starting a run).
2. The robot must start completely behind the vertical projection of the inside of the start line.

Completion: The robot must hold the can over 11 inches for 3 seconds so that the judges have time to measure.

Extra Optimization: The cans can be moved to a random circle, multiple cans can be added, empty cans can be swapped for full cans.



Back to the Drawing Board

Imagine you need to design a claw that will go out and pick up a dry erase marker or a claw that can lift a can 10 inches off the ground.

In your notebook:

- Create a claw design that could grip a marker.
- Create a claw design that could raise an object 10 inches off the ground.
- Things to think about:
 - How many servos will you use?
 - Do you need to give the servos time?
 - When should you set the servo position?

Assessment

Assessment 22: Stackerz

Setup: Use Surface-A. Place 2 empty 12oz soda cans, one in circle 5 and the other in circle 7.

Level: Advanced

Skill: Precision manipulating.

Desired Outcome: The robot will stack one can on top of the other.

Limitations:

1. All robots must be autonomous (no remote controls, wireless communication, or touching the robot after starting a run).
2. The robot must start completely behind the vertical projection of the inside of the start line.
3. The robot's drive wheels must completely leave the starting box (crossing over and no longer touching the black line marking the starting box).
4. The bottom of the top can must be touching the top of the bottom can.
5. The robot may not be touching either can at the end of the round.

Completion: When the robot places one can on top of the other can.

Extension – Move the cans to different circles, swap the empty cans for full cans.

Assessments and Rubrics



Suggestions: *Understanding* or *Group Collaboration* rubrics