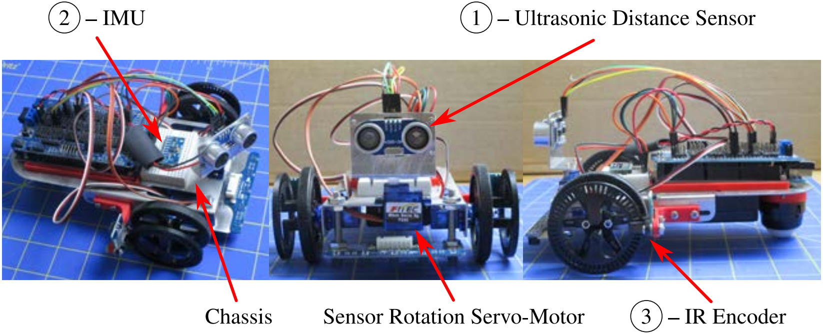
Today's Topics

- Autonomous Machines MiniBot
- Ultrasonic Distance Sensor: Operating Range
- Sensor Rotation Servo-Motor: Operating Range
- Code Functions
- Challenge for Week 1
  - Description/Rules
  - Code Functions Table
  - Code Control Structure
  - Example Code
- Start Solving the Challenge!

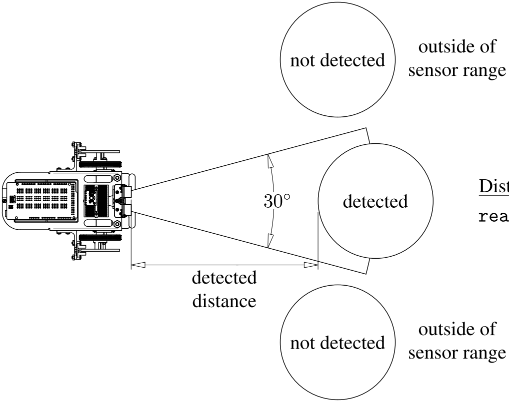
1



- MiniBot developed for 16.632 — *Introduction to Autonomous Machines*
- For your challenge, we'll use the following 3 sensors:
  1. Ultrasonic Distance Sensor (HC-SR04) — for obtaining an object's distance from the MiniBot.
  2. Inertial Measurement Unit (IMU, MPU-9250) — for determining the MiniBot's relative rotation angle.
  3. Infrared Encoder (H206)— for determining the distance traveled by the MiniBot.

Ultrasonic Distance Sensor

Operating Range



Distance Sensor Function

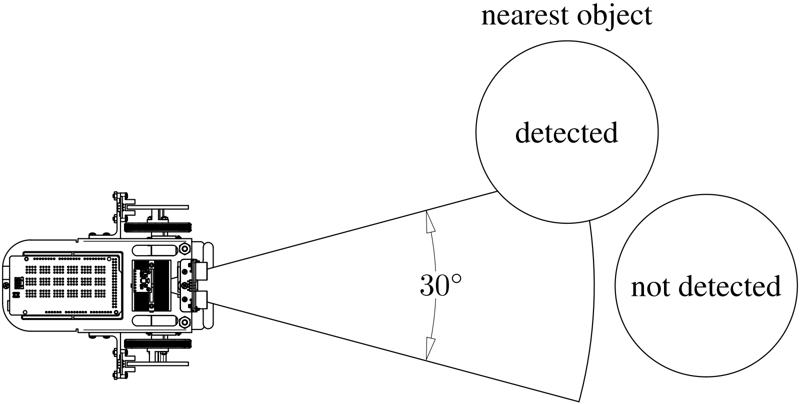
`readDistance()`

- The ultrasonic distance sensor allows the MiniBot to determine its distance away from each ball.
- Operating Distance: 2 cm (1 in) to 400 cm (13 ft)
- Resolution: 0.3 cm (0.12 in)
- Operating Angle: 30°

3

Ultrasonic Distance Sensor

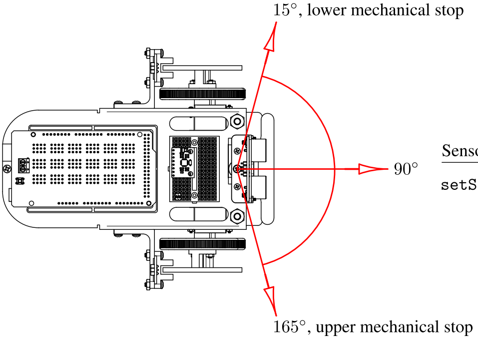
Operating Range



- If two or more objects lie within the 30° operating range, the sensor returns the distance of only the nearest object.

Sensor Rotation Servo-Motor

Operating Range



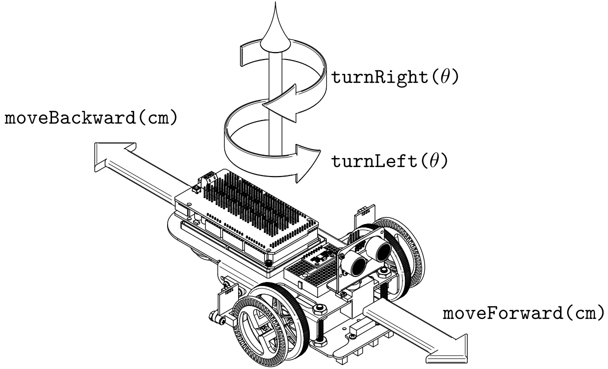
Sensor Rotation Angle Function

`setSensorAngle(`θ`)`, $15^\circ \leq \theta \leq 165^\circ$

- Sensor rotation range relative to the chassis.
- The micro-servo has a sweep range of $\sim 150^\circ$.
- You can specify your angles in increments of only 1°.

Motion Functions

IMU and Infrared Encoder Functions



- The IMU has a gyroscope, accelerometer, and magnetomoter, and it allows the MiniBot to determine its relative orientation on the gameboard.
- The IMU only accurate to within 1 degree.
- The IR sensor with encoder wheel allows the MiniBot to determine its distance traveled.

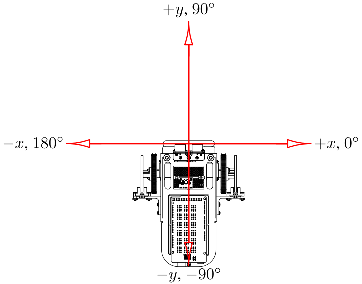
6

Location/Orientation Functions

IMU and Infrared Encoder Functions

Location/Orientation Functions

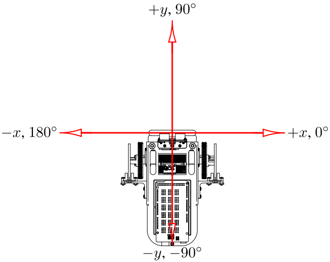
```
readXLocation()
readYLocation()
readAngle()
```



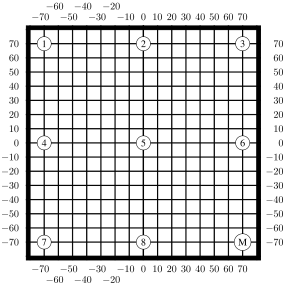
- `readXLocation()` — a function that automatically computes the ultrasonic sensor's x-location on the gameboard.
- `readYLocation()` — a function that automatically computes the ultrasonic sensor's y-location on the gameboard.
- `readAngle()` — a function that automatically computes the MiniBot's chassis orientation.

Challenge for Week 1

Description/Rules



MiniBot Starting Orientation



Possible Ball Locations

- (1) $\Longrightarrow (-70, 70)$
- (2) $\Longrightarrow (0, 70)$
- (3) $\Longrightarrow (70, 70)$
- (4) $\Longrightarrow (-70, 0)$
- (5) $\Longrightarrow (0, 0)$
- (6) $\Longrightarrow (70, 0)$
- (7) $\Longrightarrow (-70, -70)$
- (8) $\Longrightarrow (0, -70)$

MiniBot Starting Location

- (M) $\Longrightarrow (70, -70)$

1. Use the sensors' operating ranges/constraints to develop a strategy to search and find three balls placed in three distinct locations on a gameboard.
2. The gameboard contains a 160 cm by 160 cm grid spaced 10 cm apart with the origin $(0, 0)$ at the center.
3. The possible locations for the balls include only the following 8 points: $(-70, 70)$, $(0, 70)$, $(70, 70)$, $(-70, 0)$, $(0, 0)$, $(70, 0)$, $(-70, -70)$, and $(0, -70)$.

4. The MiniBot begins at the point $(70, -70)$ and oriented at $90°$ relative to the gameboard.
5. You cannot drive the MiniBot off the gameboard.
6. You must drive the MiniBot along the 10 cm gridlines. For instance, your `moveForward(cm)` and `moveBackward(cm)` commands must contain only increments of 10 cm, which means you cannot specify a command such as `moveForward(20.3)`,
7. The MiniBot cannot contact any of the balls. To prevent direct contact, you cannot drive closer than ± 20 cm of any possible ball location. For instance, you cannot drive to the locations $(0, -70)$ or $(10, -70)$, but you can drive to $(20, -70)$.
8. NOTE: if the function `readDistance()` returns a value greater than 160 cm, there are NO balls on the gameboard within the sensor's 30 deg cone operating range.

Challenge for Week 1

Code Functions

`moveForward(distance)`	`readDistance()`
`moveBackward(distance)`	`setSensorAngle(angle)`
`turnLeft(angle)`	`readAngle()`
`turnRight(angle)`	`readXLocation()`
	`readYLocation()`

- `moveForward(distance)` — moves MiniBot forward by the given distance (cm).
- `moveBackward(distance)` — moves MiniBot backward by the given distance (cm).
- `turnLeft(angle)` — turns MiniBot towards left by the given angle (degrees).
- `turnRight(angle)` — turns MiniBot towards right by the given angle (degrees).
- `readDistance()` — returns the distance of the closest object within a cone of $30°$.
- `setSensorAngle(angle)` — rotates the distance sensor to a specified angle relative to the chassis, where $15° \leq \text{angle} \leq 165°$.
- `readAngle()` — returns the orientation of MiniBot in the gameboard frame.
- `readXLocation()` — returns the x-location of MiniBot's ultrasonic sensor in the gameboard frame.
- `readYLocation()` — returns the y-location of MiniBot's ultrasonic sensor in the gameboard frame.

Challenge for Week 1

Code Control Structure: `while`

Description

A `while` loop will loop continuously, and infinitely, until the expression inside the parenthesis becomes false. When using a `while` loop, make sure to change the tested variable inside the loop, or else the loop will never exit.

Example

This example drives the MiniBot forward for 10 cm and then checks the object's distance from the ultrasonic sensor. It completes this check 3 times.

```
distanceTraveled = 0; // initialize the distance traveled
while (distanceTraveled < 30) {
  moveForward(10); // drive forward 10 centimeters
  objectDistance = readDistance(); // obtain the object's distance
  distanceTraveled = distanceTraveled + 10; // increment the distance traveled
}
```

11

Challenge for Week 1

Code Control Structure: `if ... else`

Description

The `if ... else` allows greater control over the flow of code than the basic `if` statement, by allowing multiple tests to be grouped. An `else` clause (if at all exists) will be executed if the condition in the `if` statement results in `false`.

Example

This example reads the MiniBot's x-location on the gameboard. If the x-location is less than 10 cm, then turn the MiniBot left by 90 deg; if the x-location lies between 10 cm and 20 cm, then turn the Minibot right by 90 deg; and if neither of these conditions are satisfied, then drive backward for 10 cm.

```
if (readXLocation < 10) {
   turnLeft(90);
}
else if (readXLocation <= 20) {
  turnRight(90);
}
else {
  moveBackward(10);
}
```

Challenge for Week 1

Code Control Structure: `for`

Description

The `for` statement is used to repeat a block of statements. An increment counter is usually used to increment and terminate the loop. The `for` statement is useful for any repetitive operation.

Example

This example rotates the sensor angle in 10 deg increments. It begins at 15 deg and ends at 165 deg.

```
for (angle = 15; angle <= 165; angle = angle + 10) {
    setSensorAngle(angle)
}
```

Challenge for Week 1

Code Control Structure: `return`

Description

Terminate a function and return a value from a function to the calling function, if desired.

Example

This example indicates a ball has been found; however, it does not indicate the ball's location!

```
if (readDistance() < 160) {
   return true
}
else {
  return false
}
```

14