Hide & Seek Project

Table of Contents

PHASE I.A- DEAD RECKONING W/O MOBILE ROBOT

Before you do this phase, you should have learnt the following:

- Functions
- Basic Motor encoders handling

Agent:
Program your controller to triangulate its orientation, and its distance traveled by using the encoder in the motors.

Description:

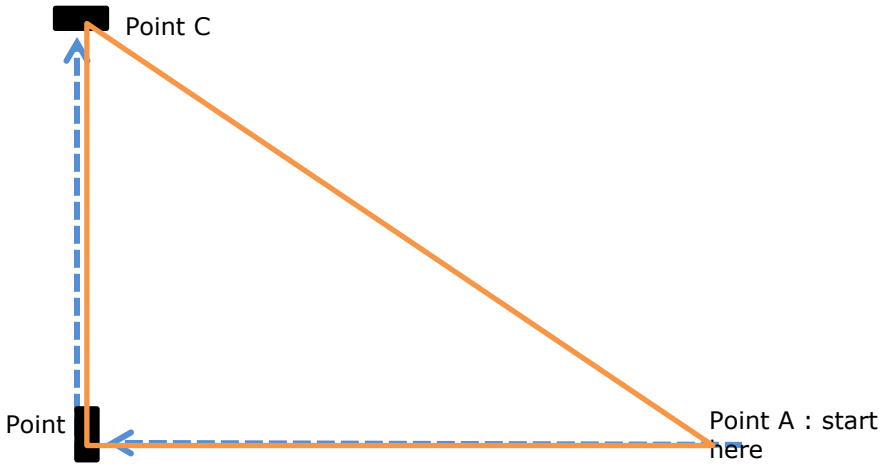
Let point A at (90, 10), point B at (5, 10), and point C at (5, 60). Imagine that a robot travels from point A to point B, then it has to figure out its way back to point A.

Your Output Display:

- Draw the following triangle on the LCD screen.
- Display the distance from point C back to A
- Display the angle rotation required to turn the orientation after reaching point C back towards point A.

Extra:
If you have time to build a robot, you can use a black tape to identify the location of Point B and C.

This robot will go from point A to B, (stopping at the black block). Right a perfect 90 degree turn, and stop at point C.
After that, it will calculate the turning degrees required to change its orientation to point A.
Then, it needs to calculate the distance it needs to move back on the point A.



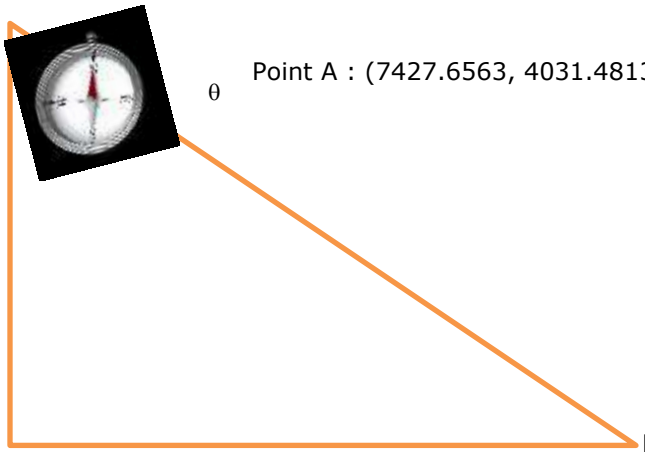
PHASE I.B - DEAD RECKONING USING LATITUDE AND LONGITUDE

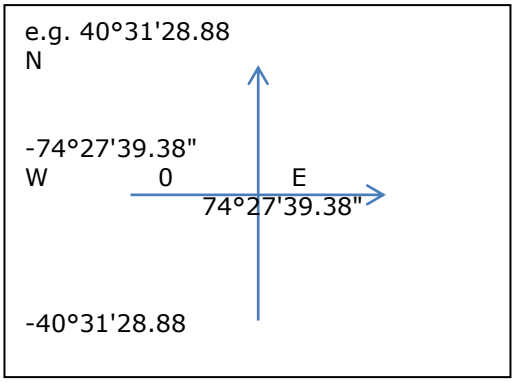
Given two locations:
Point A: Longitude = 74°27'39.38" W Latitude= 40°31'28.88"N
Point B: Longitude = 74°27'39.02"W Latitude= 40°31'28.74"N

Remember that they are represented as DDMM.mmmm, ie. 2 digits of Degree, 2 digits for Minute, and 4-decimal places for Minutes.

Therefore, the data format from your GPS may look like this:
Point A: Longitude = 7427.6563 Latitude = 4031.4813
Point B: Longitude = 7427.6503 Latitude = 4031.4790

To calculate the distance from point 1 to point 2.





Intercept X: (7427.6563, 4031.4790)

Steps:

- Take the difference. Considering you are not going to working with distance more than 10 meters away, you can simply discard the Degrees.
- Convert the distance:
- Longitude : ~= 23.33 meters per second
- Latitude : ~=33.26 meters per second (beware the this distance reduce it approaches further to N Pole, but increase to the Equator.
- ∴ BX ≅ 0.0060 * ≅ 8.64meters
- AX ≅ 0.0023 ≅ 4.59meters
- AB ≅ 9.57 meters
- $\theta \cong \tan^{-1}(\frac{8.64}{4.59})$
- $\cong 62^{o}$

PHASE II.A - REMOTE CONTROL W/ TOUCH SENSOR (WIRED)

Before you do this phase, you should have learnt the following:

- What is meant by a State Table (or somewhat like a state machine)
- Functions
- Touch Sensor Feedback

Agent:

- A single robot equipped with 3 touch sensors for navigation.
- The same robot is the engine on wheels being remotely controlled by the controller unit.

Description:

- One Controller unit will allow user to remotely drive an engine bot via Bluetooth.
- You will need to write up the states table for the remote control sequence.

Sample States Table for the controls

T1: LM T2: switch T3: RM

	T1	T2	T3
All stop	0	0	0
Left turn = LM 0% RM 75%	1	0	0
Forward = LM 75% RM 75%	1	0	1
Right Turn = LM 75% RM 0%	0	0	1
Back up = LM -75% RM -75%	0	1	0
Sharp Left = LM -75% RM 75%	1	1	0
Sharp Right = LM -75% RM -75%	0	1	1
You decide	1	1	1
Undecided!	0	1	0
Stop	0	0	0

Encoding your message:

- You need to encode your "actions message" into a single byte.

Method 1: To following the states table above to encode your action message into a single byte: example:

```
States = 0
        if  T1  pushed
                states = 100 + states
        if  T3  pushed
                states =  1 + states
        if  T2  pushed
                states = 10 + states
```

Method 2: If you know bits operation, the message encoding may look something like this :

```
States = 0
        if  T1  pushed
                states =  1<<2 | states
        if  T3  pushed
                states =  1 | states
        if  T2  pushed
                states = 1<<1  |  states
```

*** Why would I want to use bits operation? Two reasons:

- if using bits, you need only 3 bits, not 8 bits like method I. You can use the rest 5 bits for some other information;
- bits operation is faster than base10 math operations.

- After creating the action message byte is formed, write an independent function to decode this message to various actions. (Refer to the sample states table, or your states table if you design differently)

Method 1: T1state = states /100 T2state = states %100 /10 T3state = states/10

Method 2: (if you use bits operation to encode from the controller function)
 	
T1state = states & 4 // i.e. 100_2 in binary
T2state = states & 2 // ie. 10_2 in binary
T3state = states & 1 // i.e 1 in binary

Psuedo Code:

```
   byte  toEncode()  // use method 1
   {   s = 0
   if  T1  pushed
   s = 100 + s
   if  T3  pushed
   s =   1 + s
   if  T2  pushed
   s =  10 + s
   }

   task main()
   {
      Set sensors type

      While true
   {
            states = toEncode()
   th1 = states /100
   th2 = states %100 /10
   th3 = states /10
   if (!th2)
      Lm = (th1 ?  50 : 0)
      Rm = (th3 ? 50 : 0)
   Else
      Lm = (th1 ?  50 : -50)
      Rm = (th3 ? 50 : -50)
       }
}
```

PHASE II.B - REMOTE CONTROL W/ TOUCH SENSOR (WIRELESS)

Before you do this phase, you should have learnt the following:

- What is meant by a State Table (or somewhat like a state machine)
- Functions
- Touch Sensor Feedback
- Simple Bluetooth communication

Agent:

- One is the controller unit equipped with 3 touch sensors for navigation . This is a hand-held Sender/Controller unit will allow user to remotely drive an engine bot via Bluetooth
- Another robot is the Receiver/Engine on wheels being remotely controlled by the controller unit.

For the Sender/Controller:

- Check the BT connection and established stream >0.
- Create the states machine like above or use your own designed protocol.
- Encode your action message into a single byte and write it out to the BT stream.

For the Receiver/Engine:

- Check the BT stream from the receiver.
- If data stream >=0, proceed read 1 byte. This single byte will contain the encoded action message from the Controller.
- Decode it to obtain the states from T1, T2, & T3.

For the Receiver/Engine:

- After creating the action message byte is formed, write an independent function to decode this message to various actions. (Refer to the sample states table, or your states table if you design differently)

Method 1:

```
T1bit = States /100
T2bit = States %100 /10
T3bit = States/10
```

Method 2: (if you use bits operation to encode from the controller function)

```
T1bit= states & 4   // i.e. 100₂ in binary
T2bit = states & 2  // ie. 10₂ in binary
T3bit = states & 1  // i.e 1 in binary
```

- To form the movements based on the value of T1bit, T2bit, and T2bit.

```
If  T2bit ==0
        motor[RM] = (T1bit==1 ? 75 : 0);
                motor[LM] = (T3bit==1 ? 75 : 0);
else
        motor[RM] = (T1bit==1 ? 75 : -75);
                motor[LM] = (T3bit==1 ? 75 : -75);
```

Psuedo Code for Controller:

```
byte  toEncode()  // use method 1
{   s = 0
if  T1  pushed
s = 100 + s
if  T3  pushed
s =   1 + s
if  T2  pushed
s =  10 + s
}

Void checklink()  // this requires you to manually establish the link to the remote engine
{
   Bt stream >0
     Good to go
}

task main()
{
   Set sensors type
   Establish the BT stream
   While true
          states = toEncode()
send "states" data to BT stream
}
```

Psuedo Code for Engine/Receiver:

```
Void checklink()  // this requires you to manually establish the link to the remote controller
{   Bt stream >0
      Good to go
}

Void decode(byte states, byte &th1, byte &th2, byte &th3)
{
th1 = states /100
th2 = states %100 /10
th3 = states /10
}

task main()
{
   Set sensors type
   Establish the BT stream
   While true
   {
Read "states" data byte from BT stream
decode(states, th1, th2, th3)
          if (!th2)
   Lm = (th1 ?  50 : 0)
   Rm = (th3 ? 50 : 0)
Else
   Lm = (th1 ?  50 : -50)
   Rm = (th3 ? 50 : -50)
    }
}
```

PHASE III - FIND TARGET ORIENTATION WITH COMPASS

Before you do this phase, you should have learnt the following:

- Data structure, Enumerated type
- Functions
- Task
- Compass Sensor Feedback

Agent:
Only a single robot with a compass sensor

Description:

- Decide the origin, and place the robot at this particular direction.
- Wait for user to press a button.
- Log its current direction.
- User manually move the robot pointing to another direction
- Wait for user to press a button.
- robot will turn back to the original direction

Pseudo code:

```
void doTurn(int targetCS)
{

   While (diff != 0)
   {
          Diff = targetCS – curr compass
          if (abs(diff) >= 180)
     {     if (diff>0)
             Turn left  // either move the motor or display on LCD to inform reader
           else
             Turn right // either move the motor or display on LCD to inform reader
     }
     else
     {
          if (diff>0)
              Turn right// either move the motor or display on LCD to inform reader
          else
             Turn left  // either move the motor or display on LCD to inform reader

     }
   }
}

task main()
{
   initialize the compass sensor
   get the target compass value
wait for pressed  (Now turn your robot to a different orientation)

doTurn(target compass value)
}
```

Caution:

1) Due to highly granular degrees change, ie. 360 per full revolution, a single degree may mean just the bot jiggling a bit.
2) Considering the overshooting effect from fast movement. Therefore, if you want a highly efficient turning, you may want to consider change of velocity at the last few degrees closest to the target.

PHASE IV.A - GET GPS DATA

Before you do this project, you must have learnt the following:

- Data structure
- Data Array
- Functions
- Task
- GPS sentence
- Communicate via Bluetooth
- Find the right gps sentence
- Get the Latitude and Longitude
- Know the issue with large float numbers

Agent:
Only a single NXT controller with GPS receiver and compass.
This controller will act like your GPS hand-held unit to tell your orientation. Display user-friendly and concise data.

Description:
NXT Controller will continuous display your gps Latitude and Longitude.

Pseudo code

```
Need the following data:
Gps Pin number string
Gps header ,i.e "$GPGGA"
A global btData string

typedef struct {
  int deg;
  int MM;
  int mm;
  int ss;
}COORD;

typedef struct {
  COORD lat;
  COORD lng;
  int nSats;
  int fix;
  char latD;
  char lngD;
} SENTENCE;

SENTENCE MyLoc;

task talkToGPS()   // look for the right GPS sentence and get the latitude and longitude
{
    while (true)
    {
           getGPGGA();
           get the ',' 
           get the next 10 bytes ( do not need the system time field)
           get the next 9 bytes for the Latitude
           get the ',' 
           get the Direction
           get the next 10 bytes for the Longitude
           get the ',' 
           get the Direction
           get the ',' 
           get the # of fix
           get the ',' 
           get the # of Satellites
```

```
    }
    }

void getGPGGA()  // this function will not exit until it finds the proper GPS header
{
    getting a single character at a time
    if  it is '$'
          continue to check the subsequent bt bytes will match the gps header array field

    if see one character not within the sequence, start from seeking the symbol "$"
}

int getChar()  // get a single character at a time
{
    Get a character from BT data stream. One byte at a time
    If valid
           Return the character
    Return -1;
}

int getChars(int count) // get multiple characters at a time
{
    Get  "count"  characters from BT data stream.
    If valid
           Return the character
    Return -1;
}

bool parseFloat(COORD &pos, int nbytes)  // get "nbytes"character at a time, parse the DMS
{
    Get "nbytes" of data
    Parse the bt data buffer and store them in "pos" structure
    If valid
       Return true
    Return false
}


bool parseInt(int &num, int nbytes) // get "nbytes"character at a time, convert to a single int
{
    get "nbytes"character
    convert the bt data buffer to a single int
    If valid
       Return true
    Return false
}

bool parseDirection(char &dir)
{ int ct;
  wait1Msec(1);// To ensure other tasks get timeslices even if this is high priority.
  ct = getChars(2);
  if (ct!=2)
    return false;
  dir = StringGetChar(STemp,0);
  switch (dir)
  {
  case 'N':
  case 'S':
  case 'E':
  case 'W':
    break;
  default:
    return false;
  }
  return true;
}

task main()
{
```

```
    Check link to the my gps
    If bad
           exit

    Start the talkToGPS task

    while(true)
    {
           Display user friendly latitude and longitude data
    }
}
```

PHASE IV.B - HAND HELD GPS CONTROLLER

Before you do this project, you must have learnt the following:
- the list in the last section.
- Convert two DMS coordinates to distance in meters

Agent:
- Only a single NXT controller with GPS receiver and compass sensor.
- This controller will act like your GPS hand-held unit to tell you the orientation and distance to go in order to find an object at a pre-defined target location. Therefore, the user display must be designed to be able to provide enough information but short and easy to read.

Description:
- Given one target location with its Latitude and Longitude.
- NXT Controller display routing information to direct reader which direction to turn, and how far to go.

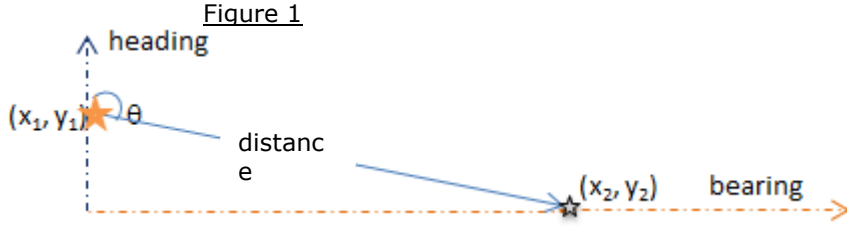
What you need to know:

Latitude == value of y-axis
Longitude == value of x-axis

Distance for Latitude: 1sec ~= 30.85 meters

Distance for Longitude: 1sec ~= 23.35 meters

Figure 1



- (x_2, y_2) is the given target
- triangulate from (x_1, y_1) to (x_2, y_2)
- Calculate the degree θ and distance in centimeters or meters.
- Display routing information to direct reader to reach the target.
- Reminder:
- X = MM.mmmmm latitude
- Y = MM.mmmmm longitude

Steps:

- Bluetooth link to the GPS (may use the existing function from the gpsRead.c
- Write a function called talkToGPS() to retrieve Latitude and Longitude information. (May use the existing function from the gpsRead.c
- Notice that 2 global variables **fLong** and **fLat** for the display.
- Wait for pressed & released
- Log your bot's location, i.e your waypoint (targeted origin):
  - create variables **targetLong** and **targetLat**
- Wait for pressed & released
  note: assume you will move the bot to a different Location about 20 feet away from original location
- At this point, you should have two set of coordinates:
  - fLong and fLat (your current position)
  - targetLong and targetLat (your target position)
- Write the Triangulate algorithm to get. (Refer to the pseudo code below)
  a. bearing towards the waypoint using the calculated compass degrees
  b. max crow-fly distance
- Call the function **turnTo(target)** that you wrote earlier to navigate the bot to bear toward the calculated compass degrees
- Repeat 8 & 9 until the **targetLong** and **targetLat** ~= **fLong** and **fLat**

(remember that GPS data can easily have about 3 meter error margin)

Pseudo code

```
void routeBack()
{

…. Now, complete your own route back pseudo code before writing the code

}

task main()
{
    Check link to the my gps
    If bad
           exit

    Wait for button pushed
    Save the gps data as target

    Start the talkToGPS task

    Wait for button pushed  // you will walk away from the target

    while(true)
    {
           routeBack();
    }

}
```

PHASE V - HIDE & SEEK (GEOCACHING SIMULATION) W/ STATIONARY TARGET

Agent:

- One search robot equipped with GPS and compass, and one hiding GPS.

Description:

- Your robot will probe the location of another GPS. This becomes your target location. Neither you nor the robot knows the location of this hidden target GPS.
- Start your another search robot
- The robot will re-route itself back to the target

PHASE VI - SEEK THE MOBILE JOKER

Agent:
Two robots:

- one batman on wheels
- one joker (does not need to be on wheels)

Each robot is equipped with its own gps receiver. The batman(sender) must also carry a compass sensor. The joker (receiver) does not need one for this phase.

General tasks for both:

1. wait for press and wait for release
2. oldDir = compass value
3. wait for press and wait for release
4. (note: assume you will move the bot to a different
5. Location about 20 feet away from receiver)
6. acquire lat & long from its own gps receiver

Batman agent role:
- constantly acquiring data from jokes's location
- find the joker by:
  - using compass to tell its (batman's) own heading
  - calculating the directional change based on joker's lat & long vs. batman's lat & long
  - need to optimize the search algorithm. One suggestion:
    a) create some sort of deviation tolerance level, ie. distance and heading vs. bearing
    b) check to see your current direction has exceeded. Also need to check distance as well.
    c) if not over the tolerance level, keep moving based on the past direction
    d) if it exceeds the tolerance level, you must recalculate and re-adjust direction.
    e) Caution: compass sensor is extremely sensitive. Therefore, before you determine the heading vs bearing exceeds the tolerance level, your robot may need to stop, allow the bot to stabilize before checking the compass again.

Joker agent role:
- carry a gps receiver and nxt
- constantly broadcasting its position
- does not move

Pseudo code:

<write your pseudo code for the search algorithm…>

Triangulate the robot's heading to change to the target direction

Connotation:

(X_n, Y_n) = (normalized Latitude, normalized Longitude)

Latitude
1sec ~= 33.26 meters

Longitude
1sec ~= 23.33 meters

$(\Delta x, \Delta y)$ = delta of any (X_n, Y_n) & (x_0, y_0)

c = $\sqrt{\Delta x^2 + \Delta y^2}$

θ = $\cos^{-1}\left(|\Delta y| / |c| \right)$

ToleranceDistance ~= 1 meter?

Assuming you have had a function *turnTo(N)* :
This is your own function that will enable your robot to turn its heading to N degrees according to Compass

e.g. turnTo(90) == turn your robot to head toward 90o compass value, ie. East.

Pseudo Code

1. Normalize the Δx & Δy
   e.g. :

```
Δx = Δx*60 * 30.9;    // convert x to seconds and then to  meters
Δy = Δy*60* 28.15;    // convert y to seconds and then to  meters
```

```
Note that the Lat & Long  are  in DDMM.mmmmmm format.  Format from most online tools, such as
Google Earth, use DD.ddddddddd format.  Therefore, if you are looking up a coordinate from
Google Earth, beware the translation.
e.g.  Latitude:         40d 31' 16.54"
      Google Earth:     40.52126111
      the $GPGGA :      4031.275667

For your application, DD will always be 0, anyway, so the normalization will consider meters per
second.
```

2. One proposed way. However, you should come up with your own.

```
    if (Δx <=  ToleranceDistance  && Δy <=   ToleranceDistance) )
                DONE!!!

if (Δy <= ToleranceDistance) // on the same longitude

          if (abs(Δx) > 0)
                turnTo( 90)
          else
                turnTo( 270)

else if (Δx <= ToleranceDistance) // on the same
latitude
          if (Δy > 0)
                turnTo( 0)
          else
                turnTo( 180)
else
    if (Δx > 0)
              if ( Δy > 0)  // 1st quad
                    turnTo(θ)
             else  if (Δy < 0)  // 4th quad
                    turnTo(180 - θ)
    else if (Δx< 0)
              if (Δy > 0)   // 2nd quad
                    turnTo(360 - θ)
             else  if (Δy < 0)  // 3rd quad
                    turnTo(180 + θ)
```