**CG2111A Engineering Principles and Practices II for CEG**
**Week 9 Studio 1**
**Alex Goes Remote**
**(PARTIALLY GRADED STUDIO)**

## INTRODUCTION

In Week 8 Studio 2 we have started to establish our control over Alex by looking at how to count the number of ticks from the wheel encoders, and how to transfer data correctly between the Pi and the Arduino.

In this studio we carry on first by modifying Alex's core firmware routines to count ticks not only moving forward, but also moving backwards, and we will now also measure how far Alex has moved backwards, not just forwards.

More excitingly though in this Studio we will learn how to remotely tell Alex how far forward/backwards to move, and even tell him what angles to turn left or right.

The activities in this studio are:

**Activity 0.** Putting in Debug Printing. (UNGRADED)
**Activity 1.** Keeping track of current direction, and forward and backward distances. **(GRADED)**
**Activity 2.** Controlling Alex remotely. **(GRADED)**
**Activity 3.** Getting telemetry from Alex. (UNGRADED)
**Activity 4.** Getting Alex to turn X degrees left or right, or moving Y meters forward or backwards. (UNGRADED)

**ONLY ACTIVITIES ONE AND TWO ARE GRADED, BUT YOU MUST COMPLETE ALL FOUR ACTIVITIES IN ORDER FOR ALEX TO FUNCTION. THIS STUDIO IS WORTH 15 MARKS.**

## TEAM SETUP

You will work in your project level teams of 4 to 5 persons.

## EQUIPMENT SETUP

You will need your completed code from Week 8 Studio 2. You also need a fully assembled Alex with ssh and VNC access.

## SUBMISSION

Rename the AxxxxxY.docx file to the student ID of the student submitting. Only one copy is required. Upload your respective studio's submission folder by 12.15 pm (morning sessions) or 5.15 pm (afternoon sessions).

## ACTIVITY 0. DEBUG PRINTING

**YOU SHOULD NEVER USE THE SERIAL LIBRARY FOR DEBUG PRINTING! DOING SO WILL BREAK THE ALEX PROTOCOL AND YOU WILL RECEIVE "BAD MAGIC NUMBER" AND OTHER ERRORS. THERE IS NO WAY TO RECOVER FROM THIS EXCEPT TO REBOOT ALEX.**

**Open Alex.ino.** For debug printing, you can add the following code to the **Alex.ino** Firmware file (i.e. the code running on Alex's Arduino).

1. At the start of Alex.ino, add in:

   #include <stdarg.h>

2. Locate the "sendMessage" function, and AFTER this function, add the following code:

```
void dbprintf(char *format, ...) {
      va_list args;
      char buffer[128];

      va_start(args, format);
      vsprintf(buffer, format, args);
      sendMessage(buffer);
}
```

You can now call "dbprintf" the same way you call printf, e.g.:

```
      #define PI  3.141592654

      dbprintf("PI is %3.2f\n", PI)
```

This will make whatever you print appear on the Alex client program running on the Pi (e.g. alex-pi in this studio). Note that if what you are printing exceeds 128 bytes in total, your Arduino will crash.

**Again you should only use dbprintf for your debug printing. DO NOT USE SERIAL.WRITE, etc.**

## ACTIVITY 1. KEEPING TRACK OF DIRECTION AND DISTANCE

In Week 8 Studio 2 we started tracking how far forward Alex has moved based on clicks registered by the wheel encoders. We didn't record how far Alex travels backwards. We will now fix this issue. There are two related problems though:

i) We only have the variables leftTicks and rightTicks to measure the number of ticks returned by the wheel encoders. We didn't keep track if these were ticks triggered moving forward or backward.

ii) The wheel encoder returns the exact same ticks whichever direction you turned the wheel. This means that based on the encoders alone, you don't know which direction Alex is heading.

We will now address these two issues:

Step 1.

Locate the line that says “volatile unsigned long leftTicks;”. Change “leftTicks” to “leftForwardTicks” and “rightTicks” to “rightForwardTicks”. Similarly add two more variables “leftReverseTicks” and “rightReverseTicks”.

We will also maintain a separate set of counters for turning. Create four more counters called leftForwardTicksTurns, leftReverseTicksTurns, rightForwardTicksTurns and rightReverseTicksTurns:

```
/*
 *    Alex's State Variables
 */

// Store the ticks from Alex's left and
// right encoders.
volatile unsigned long leftForwardTicks;
volatile unsigned long rightForwardTicks;
volatile unsigned long leftReverseTicks;
volatile unsigned long rightReverseTicks;

// Left and right encoder ticks for turning
volatile unsigned long leftForwardTicksTurns;
volatile unsigned long rightForwardTicksTurns;
volatile unsigned long leftReverseTicksTurns;
volatile unsigned long rightReverseTicksTurns;
```

Step 2.

**Open constants.h.** Edit the “constants.h” file, and add this enum just below the TCommandType enum:

```
typedef enum
{
  FORWARD=1,
  BACKWARD=2,
  LEFT=3,
  RIGHT=4
} TDirection;
```

Your code in constants.h should look like this:

```
  COMMAND_CLEAR_STATS = 6
} TCommandType;

typedef enum
{
  FORWARD=1,
  BACKWARD=2,
  LEFT=3,
  RIGHT=4
} TDirection;
```

**Open Alex.ino.** Next edit Alex.ino, and add this statement just below" the '#include "constants.h"' line:

```
volatile TDirection dir;
```

Your Alex.ino code should now look like this:

```
#include <serialize.h>

#include "packet.h"
#include "constants.h"

volatile TDirection dir;
```

**Open robotlib.ino.** There is a naming conflict in robotlib.ino, and we will now fix it. Open robotlib.ino, and rename the "dir" enum to "Tdir" as shown below:

**BEFORE:**

```
#include <AFMotor.h>
// Direction values
typedef enum dir
{
  STOP,
  GO,
  BACK,
  CCW,
  CW
} dir;
```

**AFTER:**

```
#include <AFMotor.h>
// Direction values
typedef enum Tdir
{
  STOP,
  GO,
  BACK,
  CCW,
  CW
} Tdir;
```

Now **CUT the entire Tdir declaration out from robotlib.ino**, and **paste it into constants.h**, right after the declaration for TDirection. Tdir should no longer be in robotlib.ino, and constants.h should now look like this:

```
typedef enum
{
  FORWARD=1,
  BACKWARD=2,
  LEFT=3,
  RIGHT=4
} TDirection;

// Direction values
typedef enum Tdir
{
  STOP,
  GO,
  BACK,
  CCW,
  CW
} Tdir;

#endif
```

Tdir should be completely excised from robotlib.ino. The top will now look like this:

```
#include <AFMotor.h>

// Motor control
#define FRONT_LEFT    4 // M4 on the driver shield
```

Step 3.

**Open robotlib.ino.** There are five routines called “forward()”, “backward()”, “ccw()”, “cw()” and “stop” in robotlib.ino.

**NOTE:** We will use the STOP enum from Tdir for the “stop” routine. For the remaining routines we will use the enums from TDirection.

Modify these to set the “dir” variable correctly. The “forward()” routine is shown below. The TDirection cast is technically only needed for the “stop()” routine (because the STOP enum actually comes from Tdir and not TDirection), but we will do the cast for all five routines for consistency.

```
void forward(float dist, float speed)
{
  dir = (TDirection) FORWARD;
  move(speed, FORWARD);
}
```

Conventionally, “ccw()” should turn Alex left, and “cw()” should turn Alex right. Adjust your ccw and cw code accordingly.

**Open Alex.ino.** Also add the following functions to Alex.ino (After all the #include statements):

```
void left(float ang, float speed) {
  ccw(ang, speed);
}


void right(float ang, float speed) {
  cw(ang, speed);
}
```

Step 4.

Modify the clearCounters() function to zero all the ticks and distance variables (leftForwardTicks, rightForwardTicks, leftForwardTicksTurns, forwardDist, etc.). Your function should no longer be referencing old variables like leftTicks and RightTicks as these no longer exist.

Similarly locate the clearOneCounter(int which) function, remove all the code inside the function and replace it with a call to clearCounters:

```
// Clears one particular counter
void clearOneCounter(int which)
{
  clearCounters();
}
```

In future you can modify this function to clear just one or one set of counters.

**Question 1a. (3 marks)**

Modify leftISR and rightISR to correctly update the leftForwardTicks, rightForwardTicks, leftReverseTicks and rightReverseTicks variables based on dir when dir is FORWARD or BACKWARD, but not LEFT nor RIGHT.

You should also update the turns variables (leftForwardTicksTurns etc) based on dir when dir is either LEFT or RIGHT but not FORWARD nor BACKWARD.

**Question 1b**. (4 marks)

In leftISR, when Alex is moving FORWARD update the forwardDist variable using:

```
if(dir == FORWARD)
        forwardDist = (unsigned long) ((float) leftForwardTicks / COUNTS_PER_REV * WHEEL_CIRC);
```

Similarly update reverseDist when Alex is moving BACKWARD.

Do not update these variables when Alex is turning LEFT nor when he is turning RIGHT.

(Note: we assume that when dir is FORWARD and dir is BACKWARD the number of clicks by the left and right encoders is similar, and we update forwardDist and reverseDist only in leftISR, and not in rightISR.

Cut, paste and explain your ISR code in your report for Questions 1a and 1b.

## ACTIVITY 2. CONTROLLING ALEX REMOTELY

You will now provision Alex for remote control. There are two parts to controlling Alex. The main Alex routines are in Alex.ino, while the Pi control code is in alex-pi.cpp.

Step 1.

Copy w9s2.zip to the Pi if you've not already done so. Unzip w9s2.zip to the SAME directory you created last week for Week 8 Studio 2, containing the serialize.h, serialize.cpp and packet.h files. Allow zip to overwrite any existing files (in particular allow it to overwrite serial.h and serial.cpp)

**NOTE: YOU MUST UNZIP w9s2.zip IN THE SAME DIRECTORY YOU CREATED IN WEEK 8 STUDIO 2. IN PARTICULAR THE serialize.cpp AND serialize.h FILES MUST BE PRESENT!**

Using your favourite editor open packet.h and familiarize yourself with Alex's command / status packet format:

| Field | Type | Description |
|---|---|---|
| packetType | char | Type of packet. Consult the TPacketType enum in constants.h for details. |
| command | char | Command to be executed by Alex when sent by Pi to Arduino, or Alex's status when sent by Arduino to Pi.<br><br>Consult the TCommandType enum in constants.h for details on commands sent by the Pi to the Arduino, and the TResponseType enum for details on responses sent by the Arduino to the Pi. |
| data | char [32] | 32 character string for Arduino to send back text messages to the Pi. |
| params | uint32_t[16] | Array of 16 uint32_t integers for the Pi to send command parameters to the Arduino, or for the Arduino to send back telemetry data to the Pi. |
| dummy | char[2] | Two-byte padding to align the Arduino's packets with the Pi's packets. |

**Open alex-pi.cpp.** Now open alex-pi.cpp, and look at the sendCommand and getParams functions.

**Question 2.** (2 marks)

What parameters do the Pi pass along to the Arduino when sending the COMMAND_FORWARD, COMMAND_REVERSE, COMMAND_TURN_LEFT and COMMAND_TURN_RIGHT commands? (Hint: Look at what the getParams function writes to the params array.)

Step 2.

**Open robotlib.ino.** Using the Arduino IDE on the Pi, open robotlib.ino, and look at the forward(.), reverse(.), cw(.) and ccw(.) functions.

**Question 3.** (2 marks)

What do the first parameters of these functions do? The second?

Step 3.

**Open Alex.ino.** Look at the loop() function, and in the commented out part you will see that loop reads packets from the Pi, deserializes the packets, then calls handlePacket.

The handlePacket function then checks the packetType, and calls handleCommand if the packetType is PACKET_TYPE_COMMAND.

From here you can trace the execution to handleCommand. The command handling for COMMAND_FORWARD has been done for you. Notice that the first thing the case for COMMAND_FORWARD does is to call sendOK().

**Question 4.** (2 marks)

Describe how sendOK() works, and why it is needed. Also describe what sendResponse does.

Now modify handleCommand to handle the COMMAND_REVERSE, COMMAND_TURN_LEFT, COMMAND_TURN_RIGHT and COMMAND_STOP commands. You can call the newly created left(.) and right(.) functions from Step 3 of Activity 1 above, which is clearer than using ccw(.) and cw(.).

Remember to call sendOK() each time.

**Question 5.** (2 marks)

Cut and paste the code for handling COMMAND_TURN_LEFT into your report.

Step 4.

**IMPORTANT:**

**COMMENT OUT all the Serial.print and Serial.println statements in leftISR and rightISR.**

Go to loop() and comment out the forward(0,100); statement if it is uncommented. Then uncomment the code that reads the serial port and executes the packets that come in from the Pi. You can do this by removing the /* and */. Your code will look like this:

```
// forward(0, 100);

// Uncomment the code below for Week 9 Studio 2


 // put your main code here, to run repeatedly:
  TPacket recvPacket; // This holds commands from the Pi

  TResult result = readPacket(&recvPacket);

  if(result == PACKET_OK)
    handlePacket(&recvPacket);
  else
    if(result == PACKET_BAD)
    {
      sendBadPacket();
    }
    else
      if(result == PACKET_CHECKSUM_BAD)
      {
        sendBadChecksum();
      }
```

Step 5.

Compile Alex.ino and upload to the Arduino. On the Pi, compile alex-pi.cpp:

gcc alex-pi.cpp serial.cpp serialize.cpp –pthread –o alex-pi

Now run alex-pi:

./alex-pi

Now you can control Alex!

- Type “f” followed by enter, then key in the speed (in percent) and distance to travel in cm separated by space, and press enter. E.g.

  80 50

  This means go 80 cm at 50% power.

  If all goes well, Alex will start moving forward. Indefinitely.

  Type “s” followed by enter to stop Alex.

- Similarly, you can type “b” for Alex to move backwards, “l” for Alex to turn left, and “r” for Alex to turn right.

Congratulations you now have a very crude but fun way to control Alex! Incidentally you will notice that Alex runs continuously regardless of the distance or the angle you enter. We will fix this in Activity 4.

**NOTE: THERE ARE NO MORE GRADED QUESTIONS IN ACTIVITY 3 AND 4. YOU MAY SUBMIT YOUR REPORT ONCE YOU HAVE COMPLETED ACTIVITY 2.**

**HOWEVER YOU MUST COMPLETE ACTIVITIES 3 AND 4 OR YOU WILL NOT BE ABLE TO DO YOUR FINAL PROJECT. ACTIVITIES 3 AND 4 ARE IMPORTANT!**

## ACTIVITY 3. GETTING TELEMETRY FROM ALEX

"Telemetry" – taking measurements remotely from a distance – refers to statistics that a remote vehicle or computer system gathers about itself. E.g. of telemetry include location, speed, battery voltage level, etc, to be sent to a central control station.

In this activity you will configure Alex to return all the telemetry data you've been gathering in Activity 1.

Step 1.

**Open Alex.ino.** Locate and modify the sendStatus function in this way:

- Create a new packet called statusPacket.
- Packet type should be PACKET_TYPE_RESPONSE.
- The "command" field should be set to RESP_STATUS.
- Set the params array in this way:
  0: leftForwardTicks
  1: rightForwardTicks
  2: leftReverseTicks
  3: rightReverseTicks
  4: leftForwardTicksTurns
  5: rightForwardTicksTurns
  6: leftReverseTicksTurns
  7: RightReverseTicksTurns
  8: forwardDist
  9: reverseDist

Call sendResponse to send out the packet. You can refer to some of the other send functions on how to call sendResponse.

**Question 6.** (UNGRADED)

Cut, paste and explain your code for sendStatus.

Step 2.

We will now modify handleCommand to handle the telemetry commands COMMAND_GET_STATS and COMMAND_CLEAR_STATS. For COMMAND_GET_STATS, just call the sendStatus function you created in Step 1.

COMMAND_CLEAR_STATS is slightly trickier; the alex-pi program actually specifies which counters to clear in params[0] of the command packet. Therefore in your implementation you should call clearOneCounter (which takes a parameter specifying exactly which counter to clear – but in our implementation clears all the counters), giving it command->params[0] as a parameter:

```
clearOneCounter(command->params[0]);
```

COMMAND_CLEAR_STATS should also send back an OK packet to the Pi.

**Question 7.** (UNGRADED)

Copy, paste and explain the modifications you made to handleCommand to support the telemetry commands.

We add in case statements for COMMAND_GET_STATS and COMMAND_CLEAR_STATS, shown below.

Step 3.

Ensure that alex-pi is not running on the Pi (Press 'q' to exit if it is).

Compile and upload Alex.ino to the Arduino, then on the Pi, then restart alex-pi on the Pi. Now press "g" and enter to read the telemetry off the Arduino.

## ACTIVITY 4. CONTROLLING ALEX'S DISTANCE AND TURN ANGLE

We will now embark on the most challenging part of today's studio; controlling the distance that Alex moves and the angle he turns.

We will begin with controlling the distance since this is the easier of the two:

Our algorithm works like this: We get the distance to travel, and compute the new total distance travelled. Then within main we keep checking, as Alex moves, whether we have reached this target distance. If so we stop.

Step 1.

Just after your declaration for reverseDist, enter the following lines:

```
unsigned long deltaDist;
```

```
unsigned long newDist;
```

Your code will look like this:

```
// Forward and backward distance traveled
volatile unsigned long forwardDist;
volatile unsigned long reverseDist;

// Variables to keep track of whether we've moved
// a commanded distance

unsigned long deltaDist;
unsigned long newDist;
```

Step 2.

**Open robotlib.ino.** Modify the forward() function in robotlib.ino by adding the following lines:

```
if(dist > 0)
      deltaDist = dist;
else
      deltaDist=9999999;

newDist=forwardDist + deltaDist;
```

Notice that when the specified distance given is 0, we use a very large number so that Alex continues moving indefinitely. The top part of your forward() function will now look like this:

```
void forward(float dist, float speed)
{

  if(dist > 0)
    deltaDist = dist;
  else
    deltaDist=9999999;

  newDist=forwardDist + deltaDist;

  dir = (TDirection) FORWARD;
  move(speed, FORWARD);
}
```

**Note:** If the compiler complains that deltaDist, forwardDist or newDist is an unknown variable, put the following lines at the top of robotlib.ino:

```
extern long deltaDist;
extern long newDist;
extern long forwardDist;
```

Step 3.

**Open Alex.ino.** We will now modify loop() to check for when forwardDistance now exceeds newDist as (remember that forwardDist is updated automatically by leftISR as the wheel turns). At the bottom of loop() add the following code:

```
if(deltaDist > 0)
{
      if(dir==FORWARD)
      {
            if(forwardDist > newDist)
            {
                  deltaDist=0;
                  newDist=0;
                  stop();
            }
      }
      else
            if(dir == BACKWARD)
            {
                  if(reverseDist > newDist)
                  {
                        deltaDist=0;
                        newDist=0;
                        stop();
                  }
            }
            else

if(dir == STOP)
                  {
                        deltaDist=0;
                        newDist=0;
                        stop();
                  }
}
```

Your code will look like this:

```
    sendBadChecksum();
  }

 if(deltaDist > 0)
{
 if(dir==FORWARD)
 {
  if(forwardDist > newDist)
  {
   deltaDist=0;
   newDist=0;
   stop();
  }
 }
 else
  if(dir == BACKWARD)
  {
   if(reverseDist > newDist)
   {
    deltaDist=0;
    newDist=0;
    stop();
   }
  }
  else

if(dir == STOP)
    {
     deltaDist=0;
     newDist=0;
     stop();
    }
}
```

From what you have seen in steps 1 and 2, edit reverse() accordingly to allow Alex to reverse a determined distance and stop.

Now compile Alex.ino, and upload to the Arduino. Start running alex-pi, press "f" to go forward, and enter 20 100, which should move Alex at 100% power for approximately 20 cm.

You may need to make adjustments to your code to get as close as possible to 20 cm. We leave this as an exercise for you.

Step 4.

Now we will deal with turning. This one is much more complicated. First use a ruler to measure Alex's length (longer axis) and breadth (narrower axis). Take note of these.

**Open Alex.ino.** Just after the #includes at the top of Alex.ino, define two symbols ALEX_LENGTH and ALEX_BREADTH, using the values you measured earlier. We also declare a symbol PI to be 3.141592654.

Then declare two float variables called "alexDiagonal" and "alexCirc" which will be used to estimate Alex's diagonal measurement and the circumference of his turns **(it is assumed that Alex "turns on a dime", i.e. rotates in place around the vertical axis without moving forward or backward)**. Your finished code will look similar to this:

```
// PI, for calculating circumference
#define PI               3.141592654

// Alex's length and breadth in cm. You must measure
// and substitute with the correct values.
#define ALEX_LENGTH     16
#define ALEX_BREADTH    6

// Alex's Diagonal. We compute and store this once since
// it is expensive to compute and never changes.
float alexDiagonal = 0.0;

// Alex's turning circumference, calculated once. We
// assume that Alex "turns on a dime"
float alexCirc = 0.0;
```

Just under newDist that you declared earlier on, declare the following two variables, which will be used to figure out how many wheel ticks we need to turn the left and right wheels to achieve a turn of X degrees:

```
unsigned long deltaTicks;
unsigned long targetTicks;
```

Your finished code will look like this:

```
// Variables to keep track of our turning angle
unsigned long deltaTicks;
unsigned long targetTicks;
```

Step 5.

Right at the very top of Alex.ino where all the #include statements are, enter the following:

```
#include <math.h>
```

We now modify setup() to compute Alex's diagonal length (which forms the diameter of a circle inscribed by him turning 360 degrees "on a dime"). At the top of setup() type in the following code:

```
alexDiagonal = sqrt((ALEX_LENGTH * ALEX_LENGTH) + (ALEX_BREADTH *
      ALEX_BREADTH));

alexCirc = PI  * alexDiagonal;
```

The top of your setup code will look similar to this:

```
void setup() {
  // put your setup code here, to run once:
  alexDiagonal = sqrt((ALEX_LENGTH * ALEX_LENGTH) + (ALEX_BREADTH *
    ALEX_BREADTH));

  alexCirc = PI  * alexDiagonal;
```

Step 6.

Just above the and OUTSIDE of the left() function that turns Alex left, we will create a function called computeDeltaTicks that takes and angle ang and computes an estimate of the number of wheel encoder ticks needed to turn Alex the angle specified by ang.

Our estimate will use the following assumptions:

- When Alex is moving in a straight line, he will move WHEEL_CIRC cm forward (or backward) in one wheel revolution. We assume that when Alex is moving in circles "on a dime", the wheels make the same WHEEL_CIRC cm angular distance in one revolution.
- The total number of wheel turns required to turn 360 degrees is therefore alexCirc/WHEEL_CIRC, where alexCirc is the circumference of the circle made by Alex tunring on a dime. (Once again, to "turn on a dime" means that Alex rotates about its center axis, without moving forward or backward).
- To turn ang degrees, the number of wheel turns is ang/360.0 * alexCirc/WHEEL_CIRC.
- The number of ticks is ang/360.0 * alexCirc/WHEEL_CIRC * COUNTS_PER_REV.

Based on this, type in the following function just above left():

```
// New function to estimate number of wheel ticks
// needed to turn an angle
unsigned long computeDeltaTicks(float ang)
{
  // We will assume that angular distance moved = linear distance moved in one wheel
  // revolution. This is (probably) incorrect but simplifies calculation.
  // # of wheel revs to make one full 360 turn is vincentCirc / WHEEL_CIRC
  // This is for 360 degrees. For ang degrees it will be (ang * vincentCirc) / (360 * WHEEL_CIRC)
  // To convert to ticks, we multiply by COUNTS_PER_REV.

  unsigned long ticks = (unsigned long) ((ang * vincentCirc * COUNTS_PER_REV)  / (360.0 * WHEEL_CIRC));

  return ticks;
}
```

Step 7.

Now we are ready to modify left() and right(). Type in the following code just at the start of the left() function:

```
if(ang == 0)
        deltaTicks=99999999;
else
        deltaTicks=computeDeltaTicks(ang);

targetTicks = leftReverseTicksTurns + deltaTicks;
```

The top part of the left() function will look like this:

```
void left(float ang, float speed) {

  if(ang == 0)
    deltaTicks=99999999;
  else
    deltaTicks=computeDeltaTicks(ang);

  targetTicks = leftReverseTicksTurns + deltaTicks;
```

From what you have seen for left(), make similar suitable changes to right().

Step 8.

Finally add the following to the bottom of loop(), to keep checking if leftReverseTicksTurns has exceeded targetTicks, or rightReverseTicksTurns has exceeded targetTicks, in which case we stop:

```
if(deltaTicks > 0)
{
   if(dir == LEFT)
   {
     if(leftReverseTicksTurns >= targetTicks)
     {
       deltaTicks=0;
       targetTicks = 0;
       stop();
     }
   }
   else
     if(dir == RIGHT)
     {
       if(rightReverseTicksTurns >= targetTicks)
       {
         deltaTicks=0;
         targetTicks=0;
         stop();
       }
     }
     else
       if(dir == STOP)
       {
         deltaTicks=0;
         targetTicks=0;
         stop();
       }
}
```

Quit from alex-pi, and compile and upload Alex.ino the the Arduino. Restart alex-pi, and try making 90 degree turns. You may need to make adjustments to the turning code to get exactly 90 degrees. We leave this as an exercise for you.