**RTES-6483 Project: “The Need for Speed”**

|  |  |
| --- | --- |
| Course | RTES - 6483 |
| Course Instructor | Matthew Campisi |
| Student(s) | Rituraj Singh Gahlot (rsg8554) Aradhya Alamuru (aa9405) |
|  | Embedded Project |

**Contents**1. **Gyroscope**

* 1. Initializing Gyroscope
  2. Configuring Gyroscope
  3. Sampling rate and Calibration
  4. Reading values from gyroscope

2. **LCD and Touch Screen**

* 1. splashScreenView
  2. configScreenView
  3. mainScreenView
  4. resultScreenView

3. **Speed and Distance Calculation**

* 1. Circular Queue
  2. Calculating average velocity

4. **Assumptions**

5. **Results**

**1. Gyroscope**

* 1. Initializing Gyroscope
* Constructor:  
  Creating a gyroscope object int the main will call constructor to set the SPI frequency and format

|  |
| --- |
| Gyroscope::Gyroscope(SPI \*spi\_ptr, PinName ssel) {  this->spi\_ptr = spi\_ptr;  spi\_ptr->frequency(10000000);  spi\_ptr->format(8,3);  if(ssel == NC) {  \_spi\_ssel\_ptr = NULL;  }else{  \_spi\_ssel\_ptr = new DigitalOut(ssel, 1);  }  } |

* Read and write register API’s

|  |
| --- |
| void Gyroscope::write\_register(uint8\_t reg, uint8\_t val) {  reg = reg & 0x7F;  if (\_spi\_ssel\_ptr != NULL) {  \_spi\_ssel\_ptr->write(0);  }  spi\_ptr->write(reg);  spi\_ptr->write(val);  if (\_spi\_ssel\_ptr != NULL) {  \_spi\_ssel\_ptr->write(1);  }  }  uint8\_t Gyroscope::read\_register(uint8\_t reg) {  uint8\_t val;  reg = reg | 0x80;  if (\_spi\_ssel\_ptr != NULL) {  \_spi\_ssel\_ptr->write(0);  }  spi\_ptr->write(reg);  val = spi\_ptr->write(0x00);  if (\_spi\_ssel\_ptr != NULL) {  \_spi\_ssel\_ptr->write(1);  }  return val;  } |

1.2 Configuring Gyroscope

* CTRL\_REG1(20h): Value set to 0x0F  
  0x00 🡪 Bandwidth Selection 00

0x00 🡪 Data rate bit 00 to set output data rate to 95Hz

0x0F🡪 PD mode set to 1 : Normal mode  
 Z-axis enable to 1  
 Y-axis enable to 1  
 X-axis enable to 1

* CTRL\_REG2(21h): Value set to 0x00  
  HPM1-HPM0 🡪 00 : Normal Mode  
  HPCF[3:0] 🡪 0000: Normal mode
* CTRL\_REG3(22h): Value set to 0x00 (Disable interrupts)
* CTRL\_REG4(23h): Value set to 0x00  
  FS1-FS0: Full scale selection set to 250dps  
  SIM: SPI serial interface mode selection mode to 4-wire
* CTRL\_REG5(24h): Value set to 0x03  
  Out\_Sel[1:0]: 11 : Out\_Sel0 and Out\_Sel1 enable  
  Hpen: 0: High-pass filter disable  
  FIFO\_EN: 0: FIFO disable  
  BOOT: 0 : Normal mode

|  |
| --- |
| int Gyroscope::init() {  front = -1, rear = -1;  is\_queue\_full = 0;  average\_angular\_velocity = 0;  queue\_count = 0;  zero\_count = 0;  uint8\_t id;  for (int i = 0; i < 3; i++) {  id = read\_register(WHO\_AM\_I\_ADDR);  if (id == Gyroscope\_ID) {  break;  }  }  if(id != Gyroscope\_ID){  return -1;  }  update\_register(CTRL\_REG4\_ADDR, 0x00, 0xC0);  update\_register(CTRL\_REG5\_ADDR, 0x03, 0x03);  update\_register(CTRL\_REG3\_ADDR, 0x00, 0x0F); //Disable interrupts  update\_register(CTRL\_REG5\_ADDR, 0x00, 0x40); //Disable FIFO  update\_register(FIFO\_CTRL\_REG\_ADDR, 0x00, 0xE0); // configure FIFO bypass mode  update\_register(CTRL\_REG4\_ADDR, 0x00, 0x30); //Set Full Scale 250  update\_register(CTRL\_REG5\_ADDR, 0, 0x10); //High Pass filter disable  update\_register(CTRL\_REG2\_ADDR, 0, 0x0F); //Disable HPF mode config  update\_register(CTRL\_REG1\_ADDR, 0, 0x30); //Bandwidth Selection 00  update\_register(CTRL\_REG1\_ADDR, 0, 0xC0); //Output Data rate ODR :95hz i.e 0  update\_register(CTRL\_REG1\_ADDR, 0x0F, 0x0F); //Enable gyroscope ; power up  return MBED\_SUCCESS;  } |

1.3 Sampling Rate and Calibration

* Sampling rate is set to 100Hz
* Sensitivity factor is set to 8.75 mdps/digit
* ZERO\_RATE\_NOISE set to 0.28, anything below this value is discarded for average velocity calculation

|  |
| --- |
| wait\_us(10000);  gyroscope.read\_data\_16(w\_dps);    gyroX = w\_dps[0] \* 0.00875f;  gyroY = w\_dps[1] \* 0.00875f;  gyroZ = w\_dps[2] \* 0.00875f;  gyroX = gyroX + ZERO\_RATE\_NOISE;  if (gyroX < 0.30) {  gyroX = 0;  } |

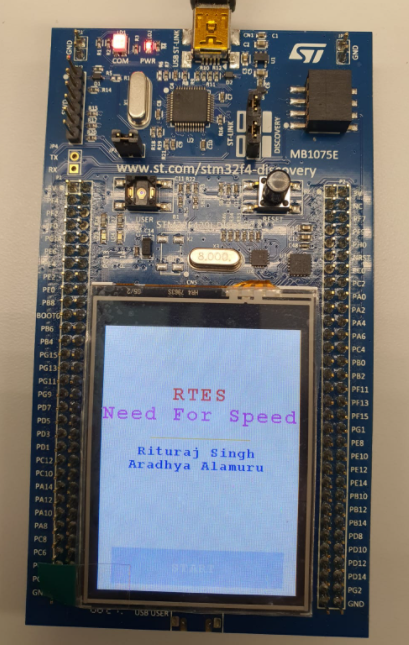
1.4 Reading values from Gyroscope

* Read\_data\_16 will read 8-bit low value, auto increment the address and then reads the 8-bit high value
* 8-bit low value is left shifted and added with 8-bit High value to create 16-bit data each from x, y and z axis

|  |
| --- |
| void Gyroscope::read\_data\_16(int16\_t data[3]) {  uint8\_t raw\_data[6];  read\_registers(OUT\_X\_L\_ADDR, raw\_data, 6);  data[0] = (int16\_t)(raw\_data[1] << 8) + raw\_data[0];  data[1] = (int16\_t)(raw\_data[3] << 8) + raw\_data[2];  data[2] = (int16\_t)(raw\_data[5] << 8) + raw\_data[4];  } |

**2. LCD and Touch Screen**2.1 splashScreenView  
 Main screen to display Project name with Start button

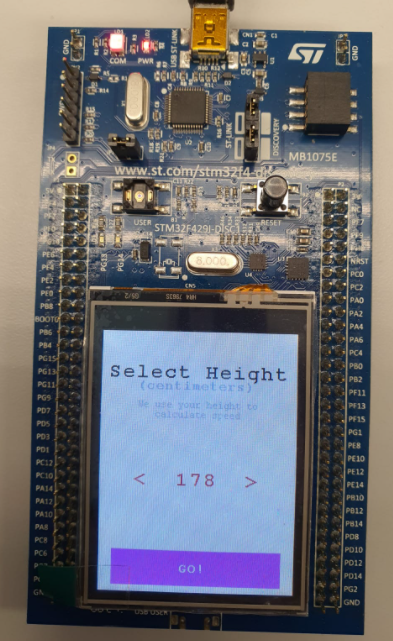
|  |
| --- |
| uint32\_t splashScreenView(uint32\_t screenWidth,uint32\_t screenHeight) {  uint32\_t activeScreen;  lcd.Clear(LCD\_COLOR\_WHITE);  lcd.SetBackColor(LCD\_COLOR\_WHITE);    wait();  lcd.SetTextColor(LCD\_COLOR\_RED);  lcd.SetFont(&Font24);  lcd.DisplayStringAt(0, LINE(3), (uint8\_t \*)"RTES", CENTER\_MODE);  lcd.SetTextColor(LCD\_COLOR\_MAGENTA);  lcd.DisplayStringAt(0, LINE(4), (uint8\_t \*)"Need For Speed", CENTER\_MODE);  wait();  lcd.SetTextColor(LCD\_COLOR\_ORANGE);  lcd.DrawLine(screenWidth / 4, 134, (screenWidth / 4) \* 3, 134);  lcd.SetFont(&Font16);  lcd.SetTextColor(LCD\_COLOR\_BLUE);  lcd.DisplayStringAt(0, LINE(9), (uint8\_t \*)"Rituraj Singh", CENTER\_MODE);  lcd.DisplayStringAt(0, LINE(10), (uint8\_t \*)"Aradhya Alamuru", CENTER\_MODE);  wait();    lcd.SetTextColor(LCD\_COLOR\_LIGHTBLUE);  lcd.SetBackColor(LCD\_COLOR\_LIGHTBLUE);  lcd.FillRect(screenMargin, screenHeight - 52, screenWidth - (2 \* screenMargin), 44);  lcd.SetTextColor(LCD\_COLOR\_WHITE);  lcd.SetFont(&Font16);  lcd.DisplayStringAt(0, screenHeight - 36, (uint8\_t \*)"START", CENTER\_MODE);  while(1) {    ts.GetState(&TS\_State);  touchX = TS\_State.X;  touchY = TS\_State.Y;  printf("x - %d, y - %d\n", touchX, touchY);  if ((TS\_State.TouchDetected) && (touchY < 40)) {  activeScreen = CONFIG\_SCREEN;  break;  }    }  return activeScreen;  } |



2.2 configScreenView

This screen will take input height from the user (Input height will be used in average velocity calculation)

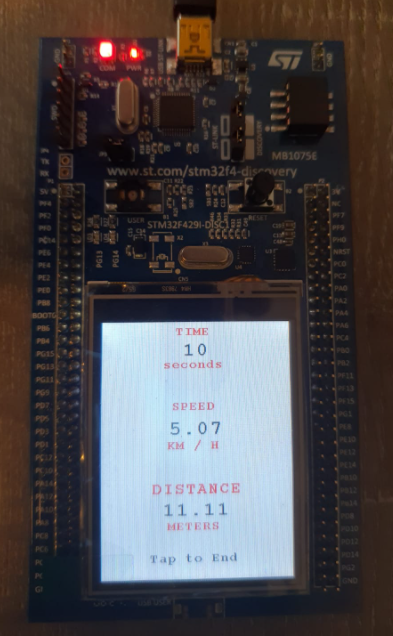
|  |
| --- |
| **uint32\_t configScreenView(uint32\_t screenWidth,uint32\_t screenHeight) {**  **uint32\_t activeScreen;**  **userHeight = 178;**  **lcd.Clear(LCD\_COLOR\_WHITE);**  **lcd.SetBackColor(LCD\_COLOR\_WHITE);**    **wait();**  **lcd.SetTextColor(LCD\_COLOR\_BLACK);**  **lcd.SetFont(&Font24);**  **lcd.DisplayStringAt(0, 40, (uint8\_t \*)"Select Height", CENTER\_MODE);**  **lcd.SetFont(&Font16);**  **lcd.SetTextColor(LCD\_COLOR\_GRAY);**  **lcd.DisplayStringAt(0, 60, (uint8\_t \*)"(centimeters)", CENTER\_MODE);**    **lcd.SetFont(&Font12);**  **lcd.DisplayStringAt(0, LINE(7), (uint8\_t \*)"We use your height to", CENTER\_MODE);**  **lcd.DisplayStringAt(0, LINE(8), (uint8\_t \*)"calculate speed", CENTER\_MODE);**  **lcd.SetTextColor(LCD\_COLOR\_DARKMAGENTA);**  **lcd.SetBackColor(LCD\_COLOR\_DARKMAGENTA);**  **lcd.FillRect(screenMargin, screenHeight - 52, screenWidth - (2 \* screenMargin), 44);**  **lcd.SetTextColor(LCD\_COLOR\_WHITE);**  **lcd.SetFont(&Font16);**  **lcd.DisplayStringAt(0, screenHeight - 36, (uint8\_t \*)"GO!", CENTER\_MODE);**    **lcd.SetTextColor(LCD\_COLOR\_DARKRED);**  **lcd.SetBackColor(LCD\_COLOR\_WHITE);**  **lcd.SetFont(&Font24);**    **while(1) {**  **sprintf((char\*)text1, " < %0.1f > ", userHeight);**  **lcd.DisplayStringAt(0, 170, (uint8\_t \*)&text1, CENTER\_MODE);**  **ts.GetState(&TS\_State);**  **touchX = TS\_State.X;**  **touchY = TS\_State.Y;**  **printf("x - %d, y - %d\n", touchX, touchY);**  **if (TS\_State.TouchDetected) {**  **if (touchY < 40) {**  **activeScreen = MAIN\_SCREEN;**  **break;**  **}**  **if ((touchY > 80) && (touchY < 160)) {**  **if (touchX > 160) {**  **increaseHeight();**  **// wait();**  **}**  **if (touchX < 80) {**  **decreaseHeight();**  **// wait();**  **}**  **}**  **}**  **}**  **return activeScreen;**  **}** |



2.3 mainScreenView

* This screen will display Time elapsed, Speed and distance travelled
* Ticker is used to display speed, time and distance after every 1 second
* Gyroscope’s x, y and z-axis angular velocities is read in this API

|  |
| --- |
| **void mainScreenTickerFunc() {**  **led != led;**  **speed = avg\_speed \* ARC\_length;**  **dist += speed;**  **lcd.SetFont(&Font24);**  **// time elapsed; seconds**  **sprintf((char\*)text1, "%d", cnt);**  **lcd.DisplayStringAt(0, 26, (uint8\_t \*)&text1, CENTER\_MODE);**  **// speed; m/s**  **sprintf((char\*)text2, " %0.2f ", speed \* 3.6);**  **lcd.DisplayStringAt(0, 124, (uint8\_t \*)&text2, CENTER\_MODE);**  **// distance; meters**  **sprintf((char\*)text3, "%0.2f", dist);**  **lcd.DisplayStringAt(0, 224, (uint8\_t \*)&text3, CENTER\_MODE);**  **cnt++;**  **}**  **uint32\_t mainScreenView(uint32\_t screenWidth,uint32\_t screenHeight) {**  **uint32\_t activeScreen;**    **speed = 0;**  **dist = 0;**  **avg\_speed = 0;**  **cnt = 1;**  **lcd.Clear(LCD\_COLOR\_WHITE);**  **lcd.SetBackColor(LCD\_COLOR\_WHITE);**  **lcd.SetTextColor(LCD\_COLOR\_RED);**  **lcd.SetFont(&Font16);**  **lcd.DisplayStringAt(0, 6, (uint8\_t \*)"TIME", CENTER\_MODE);**  **lcd.SetFont(&Font16);**  **lcd.DisplayStringAt(0, 48, (uint8\_t \*)"seconds", CENTER\_MODE);**    **// lcd.SetTextColor(LCD\_COLOR\_DARKGREEN);**  **// lcd.FillRect((screenWidth / 4) - 20, 80, (screenWidth / 2) + 40, (screenWidth / 4) + 20);**  **lcd.SetTextColor(LCD\_COLOR\_RED);**  **lcd.SetFont(&Font16);**  **lcd.DisplayStringAt(0, 100, (uint8\_t \*)"SPEED", CENTER\_MODE);**  **lcd.SetFont(&Font16);**  **lcd.DisplayStringAt(0, 148, (uint8\_t \*)"KM / H", CENTER\_MODE);**    **// lcd.SetTextColor(LCD\_COLOR\_DARKCYAN);**  **// lcd.FillEllipse(screenWidth / 2, 200, screenWidth / 4, screenWidth / 8);**  **lcd.SetTextColor(LCD\_COLOR\_RED);**  **lcd.SetFont(&Font20);**  **lcd.DisplayStringAt(0, 200, (uint8\_t \*)"DISTANCE", CENTER\_MODE);**  **lcd.SetFont(&Font16);**  **lcd.DisplayStringAt(0, 248, (uint8\_t \*)"METERS", CENTER\_MODE);**    **lcd.SetTextColor(LCD\_COLOR\_BLACK);**  **lcd.SetFont(&Font16);**  **lcd.DisplayStringAt(0, screenHeight - 32, (uint8\_t \*)"Tap to End", CENTER\_MODE);**        **int err = gyroscope.init();**  **int16\_t w\_dps[3];**  **int count = 0;**    **// button.fall(&Gyro\_On);**  **wait\_us(1000000);**  **ARC\_length = userHeight/200;**  **tick.attach(&mainScreenTickerFunc, 1.0);**    **while(1) {**  **wait\_us(100000);**  **gyroscope.read\_data\_16(w\_dps);**    **gyroX = w\_dps[0] \* 0.00875f;**  **gyroY = w\_dps[1] \* 0.00875f;**  **gyroZ = w\_dps[2] \* 0.00875f;**  **gyroX = gyroX + ZERO\_RATE\_NOISE;**  **if (gyroX < 0.30) {**  **gyroX = 0;**  **}**  **gyroX1 = gyroX \* 0.0174533f;**    **avg\_speed = gyroscope.average\_Velocity(gyroX1); //Circular queue to calculate average velocity**  **//float x1 = x \* 0.0174533f;s**    **//printf("%04d | wx1: %f, wx: %f , temp: %d\n", count, x1,x, temp);**  **printf("wx:%f wx1:%f temp:%f\n", gyroX, gyroX1, avg\_speed);**  **//printf("%04d | wx: %+7.2f dps, wy: %+7.2f dps, wz: %+7.2f dps\n", count, x, y, z);**  **//led = !led;**  **count++;**  **// touchscreen stuff**  **ts.GetState(&TS\_State);**  **touchX = TS\_State.X;**  **touchY = TS\_State.Y;**  **printf("x - %d, y - %d\n", touchX, touchY);**  **if ((TS\_State.TouchDetected) && (touchY < 40)) {**  **activeScreen = RESULT\_SCREEN;**  **break;**  **}**  **}**  **return activeScreen;**  **}** |



2.4 resultScreenView

* This screen will display total distance travelled and time for complete journey
* “TRY AGAIN” button will reset the LCD and gyroscope and will take you back to the first screen

|  |
| --- |
| uint32\_t resultScreenView(uint32\_t screenWidth,uint32\_t screenHeight) {  uint32\_t activeScreen;  lcd.Clear(LCD\_COLOR\_WHITE);  lcd.SetBackColor(LCD\_COLOR\_WHITE);    // wait();  lcd.SetTextColor(LCD\_COLOR\_DARKGREEN);  lcd.SetFont(&Font24);  lcd.DisplayStringAt(0, 40, (uint8\_t \*)"Done!", CENTER\_MODE);  lcd.SetTextColor(LCD\_COLOR\_ORANGE);  lcd.DrawLine(screenWidth / 4, 72, (screenWidth / 4) \* 3, 72);    wait();  lcd.SetTextColor(LCD\_COLOR\_BLUE);  lcd.SetFont(&Font16);  lcd.DisplayStringAt(0, 100, (uint8\_t \*)"Distance", CENTER\_MODE);  lcd.DisplayStringAt(0, 120, (uint8\_t \*)"Travelled", CENTER\_MODE);  wait();  lcd.SetFont(&Font24);  lcd.SetTextColor(LCD\_COLOR\_RED);  sprintf((char\*)text1, "%0.2f meters", dist);  lcd.DisplayStringAt(0, 137, (uint8\_t \*)&text1, CENTER\_MODE);  wait();  lcd.SetFont(&Font16);  lcd.SetTextColor(LCD\_COLOR\_BROWN);  lcd.DisplayStringAt(0, 190, (uint8\_t \*)"in", CENTER\_MODE);  lcd.SetTextColor(LCD\_COLOR\_MAGENTA);  sprintf((char\*)text2, "%d seconds", cnt);  lcd.DisplayStringAt(0, 205, (uint8\_t \*)&text2, CENTER\_MODE);  wait();  lcd.SetTextColor(0xFF57068C);  lcd.FillRect(screenMargin, screenHeight - 52, screenWidth - (2 \* screenMargin), 44);  lcd.SetBackColor(0xFF57068C);  lcd.SetTextColor(LCD\_COLOR\_WHITE);  lcd.SetFont(&Font16);  lcd.DisplayStringAt(0, screenHeight - 36, (uint8\_t \*)"TRY AGAIN", CENTER\_MODE);    while(1) {  ts.GetState(&TS\_State);  touchX = TS\_State.X;  touchY = TS\_State.Y;  printf("x - %d, y - %d\n", touchX, touchY);  if ((TS\_State.TouchDetected) && (touchY < 40)) {  activeScreen = SPLASH\_SCREEN;  break;  }  }  return activeScreen;  } |



**3. Speed and Distance Calculation**3.1 Circular Queue

* Circular queue of size 50 is implemented to capture 50 samples from gyroscope at run time at a rate of 100 samples per second
* Latest samples are En-queued and old samples are De-queued if queue is full
* A moving average is calculated at run time after each sample is read from the gyroscope. This moving average is used for calculating average velocity

|  |
| --- |
| void Gyroscope::push(float angular\_x)  {  if(angular\_x > 0){  if(queue\_count >= size && is\_queue\_full == 0)  {  is\_queue\_full = 1;  front = 0;  }  if(!is\_queue\_full)  {  rear = rear + 1;  queue[rear] = angular\_x;  average\_angular\_velocity = ((average\_angular\_velocity \* queue\_count) + angular\_x)/(queue\_count+1);  queue\_count++;  printf("1 : vel:%f qcnt:%d\n",average\_angular\_velocity,queue\_count);  }  else{  average\_angular\_velocity = ((average\_angular\_velocity \* queue\_count) - queue[front])/(queue\_count-1);  rear = front;  front = (front + 1) % size;  queue[rear] = angular\_x;  average\_angular\_velocity = ((average\_angular\_velocity \* (queue\_count-1)) + angular\_x)/(queue\_count);  printf("2 : vel:%f qcnt:%d\n",average\_angular\_velocity,queue\_count);  }  }  } |

3.2 Calculating average velocity and Distance

* Gyroscope values are converted to radian/second using factor 0.0174533f
* This value is stored in circular queue and moving average is calculated after each sample is read from the gyroscope
* Moving average angular velocity is multiplied with leg length of the user to calculate linear velocity
* This linear velocity is used in ticker function with timer to calculate total speed and distance

|  |
| --- |
| void mainScreenTickerFunc() {  led != led;  speed = avg\_speed \* ARC\_length;  dist += speed;  lcd.SetFont(&Font24);  // time elapsed; seconds  sprintf((char\*)text1, "%d", cnt);  lcd.DisplayStringAt(0, 26, (uint8\_t \*)&text1, CENTER\_MODE);  // speed; m/s  sprintf((char\*)text2, " %0.2f ", speed \* 3.6);  lcd.DisplayStringAt(0, 124, (uint8\_t \*)&text2, CENTER\_MODE);  // distance; meters  sprintf((char\*)text3, "%0.2f", dist);  lcd.DisplayStringAt(0, 224, (uint8\_t \*)&text3, CENTER\_MODE);  cnt++;  } |

4. Assumptions

* ZERO\_RATE\_NOISE is set to 0.2800, gyroscope reading below this is discarded
* Length of the leg is assumed to be half of the height of the user wearing the device
* If gyroscope reading is 0 for 30 consecutive samples, It is assumed user has stopped walking or running and moving average velocity is set to 0