Listing 1: Constants header. Currently used for UART protocol. (constants.h)

```c
#ifndef _CONSTANTS_H
#define _CONSTANTS_H

/* ==============================MC -> Pi UART Protocol===================== */

    /* Accelerometer data */
    #define ACC_X 0x20
    #define ACC_Y 0x21
    #define ACC_Z 0x22

    /* Gyroscope data */
    #define GYR_X 0x23
    #define GYR_Y 0x24
    #define GYR_Z 0x25

    /* Magnetometer data */
    #define MAG_X 0x26
    #define MAG_Y 0x27
    #define MAG_Z 0x28

    /* Altitude data */
    #define IR 0x29
    #define USONIC 0x2A

    /* Temperature data */
    #define TEMP_BAT 0x2B
    #define TEMP_M0 0x2C
    #define TEMP_M1 0x2D
    #define TEMP_M2 0x2E
    #define TEMP_M3 0x2F
    #define TEMP_AIR 0x30

    /* GPS data */
    #define GPS_N 0x31
    #define GPS_E 0x32
    #define GPS_A 0x33

    #define RECV_MAX 0x33

/* ========================================================================== */

/* ==============================Pi -> MC UART Protocol===================== */

    #define YAW_LEFT 0x20
    #define THROTTLE_UP 0x21

    #define ROLL_LEFT 0x22
    #define PITCH_FORWARD 0x23

    #define X_BUTTON 0x24

/* ========================================================================== */

#endif
```

Listing 2: Joystick++ header (rpi/joystick.hh)

```cpp
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
//
```

```
// Copyright Drew Noakes 2013-2016

#ifndef __JOYSTICK_HH__
#define __JOYSTICK_HH__

#include <string>
#include <iostream>

#define JS_EVENT_BUTTON 0x01 // button pressed/released
#define JS_EVENT_AXIS 0x02 // joystick moved
#define JS_EVENT_INIT 0x80 // initial state of device

/**
 * Encapsulates all data relevant to a sampled joystick event.
 */
class JoystickEvent
{
public:
  /** Minimum value of axes range */
  static const int16_t MIN_AXES_VALUE = -32768;

  /** Minimum value of axes range */
  static const int16_t MAX_AXES_VALUE = 32767;

  /**
   * The timestamp of the event, in milliseconds.
   */
  unsigned int time;

  /**
   * The value associated with this joystick event.
   * For buttons this will be either 1 (down) or 0 (up).
   * For axes, this will range between MIN_AXES_VALUE and MAX_AXES_VALUE.
   */
  int16_t value;

  /**
   * The event type.
   */
  unsigned char type;

  /**
   * The axis/button number.
   */
  unsigned char number;

  /**
   * Returns true if this event is the result of a button press.
   */
  bool isButton()
  {
    return (type & JS_EVENT_BUTTON) != 0;
  }

  /**
   * Returns true if this event is the result of an axis movement.
   */
  bool isAxis()
  {
    return (type & JS_EVENT_AXIS) != 0;
  }

  /**
   * Returns true if this event is part of the initial state obtained when
   * the joystick is first connected to.
   */
  bool isInitialState()
  {
    return (type & JS_EVENT_INIT) != 0;
  }

  /**
   * The ostream inserter needs to be a friend so it can access the
   * internal data structures.
   */
  friend std::ostream& operator<<(std::ostream& os, const JoystickEvent& e);
};

/**
 * Stream insertion function so you can do this:
 *    cout << event << endl;
 */
std::ostream& operator<<(std::ostream& os, const JoystickEvent& e);

/**
 * Represents a joystick device. Allows data to be sampled from it.
 */
class Joystick
{
private:
  void openPath(std::string devicePath, bool blocking=false);

  int _fd;

public:
  ~Joystick();

  /**
   * Initialises an instance for the first joystick: /dev/input/js0
   */
  Joystick();

  /**
   * Initialises an instance for the joystick with the specified,
   * zero-indexed number.
   */
  Joystick(int joystickNumber);

  /**
   * Initialises an instance for the joystick device specified.
   */
  Joystick(std::string devicePath);

  /**
   * Initialises an instance for the joystick device specified and provide
   * the option of blocking I/O.
   */
  Joystick(std::string devicePath, bool blocking);

  /**
   * Returns true if the joystick was found and may be used, otherwise false.
   */
  bool isFound();

  /**
   * Attempts to populate the provided JoystickEvent instance with data
   * from the joystick. Returns true if data is available, otherwise false.
   */
  bool sample(JoystickEvent* event);
};

#endif
```

Listing 3: Joystick++ (rpi/joystick.cc)

```
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
```

```cpp
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
//
// Copyright Drew Noakes 2013-2016

#include "joystick.hh"

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <iostream>
#include <string>
#include <sstream>
#include "unistd.h"

Joystick::Joystick()
{
  openPath("/dev/input/js0");
}

Joystick::Joystick(int joystickNumber)
{
  std::stringstream sstm;
  sstm << "/dev/input/js" << joystickNumber;
  openPath(sstm.str());
}

Joystick::Joystick(std::string devicePath)
{
  openPath(devicePath);
}

Joystick::Joystick(std::string devicePath, bool blocking)
{
  openPath(devicePath, blocking);
}

void Joystick::openPath(std::string devicePath, bool blocking)
{
  // Open the device using either blocking or non-blocking
  _fd = open(devicePath.c_str(), blocking ? O_RDONLY : O_RDONLY | O_NONBLOCK);
}

bool Joystick::sample(JoystickEvent* event)
{
  int bytes = read(_fd, event, sizeof(*event));

  if (bytes == -1)
    return false;

  // NOTE if this condition is not met, we’re probably out of sync and this
  // Joystick instance is likely unusable
  return bytes == sizeof(*event);
}

bool Joystick::isFound()
{
  return _fd >= 0;
}

Joystick::~Joystick()
{
  close(_fd);
}

std::ostream& operator<<(std::ostream& os, const JoystickEvent& e)
{
  os << "type=" << static_cast<int>(e.type)
```

```
    << " number=" << static_cast<int>(e.number)
    << " value=" << static_cast<int>(e.value);
  return os;
}
```

Listing 4: Type definition for sensor data (rpi/sensorData.hh)

```
#ifndef _SD_H
#define _SD_H

typedef struct
{
  int16_t acc_x;
  int16_t acc_y;
  int16_t acc_z;

  int16_t gyr_x;
  int16_t gyr_y;
  int16_t gyr_z;

  int16_t mag_x;
  int16_t mag_y;
  int16_t mag_z;

  int16_t ir;
  int16_t usonic;

  int16_t temp_bat;
  int16_t temp_m0;
  int16_t temp_m1;
  int16_t temp_m2;
  int16_t temp_m3;
  int16_t temp_air;

  int16_t gps_n;
  int16_t gps_e;
  int16_t gps_a;
} sensordata_t;

#endif
```

Listing 5: Function for formatting incoming sensor data. (rpi/sensorData.cc)

```
#include "sensorData.h"

int formatData(sensordata_t* sensorData, char* rawdata)
{
    tempData = (rawdata[1] << 8) | rawdata[2];
    switch (rawdata[0])
    {
        case ACC_X:
            sensorData->acc_x = tempData;
            break;
        case ACC_Y:
            sensorData->acc_y = tempData;
            break;
        case ACC_Z:
            sensorData->acc_z = tempData;
            break;

        case GYR_X:
            sensorData->gyr_x = tempData;
            break;
        case GYR_Y:
            sensorData->gyr_y = tempData;
            break;
        case GYR_Z:
            sensorData->gyr_z = tempData;
            break;

        case MAG_X:
            sensorData->mag_x = tempData;
            break;
        case MAG_Y:
            sensorData->mag_y = tempData;
            break;
        case MAG_Z:
            sensorData->mag_z = tempData;
            break;

        case IR:
            sensorData->ir = tempData;
            break;
        case USONIC:
            sensorData->usonic = tempData;

        case TEMP_BAT:
            sensorData->temp_bat = tempData;
            break;
        case TEMP_M0:
            sensorData->temp_m0 = tempData;
            break;
        case TEMP_M1:
            sensorData->temp_m1 = tempData;
            break;
        case TEMP_M2:
            sensorData->temp_m2 = tempData;
            break;
        case TEMP_M3:
            sensorData->temp_m3 = tempData;
            break;
        case TEMP_AIR:
            sensorData->temp_air = tempData;
            break;

        case GPS_X:
            sensorData->gps_n = tempData;
            break;
        case GPS_Y:
            sensorData->gps_e = tempData;
            break;
        case GPS_Z:
            sensorData->gps_a = tempData;
            break;

        default:
            printf("Unknown control byte: 0x%x\n", rawdata[0]);
            printf("\tPayload: 0x%x 0x%x\n", rawdata[1], rawdata[2]);
            return 1;
    }
    return 0;
}
```

Listing 6: Header file for RPi comms library. (rpi/mc_comms.hh)

```
#ifndef _MC_COMMS_H
#define _MC_COMMS_H

#include <stdint.h>

#define SERIAL_DEVICE "/dev/serial0"
#define RECVBUFFER_SIZE 5
#define SENDBUFFER_SIZE 5

FILE* uartInit(const char* device);
void uartClose(FILE* uartDevice);
char* uartReadRaw(FILE* uartDevice);
int uartSendRaw(char* string, FILE* uartDevice);
int uartSendCommand(uint8_t command, int16_t data, FILE* uartDevice);

#endif
```

Listing 7: RPi comms library. (rpi/mc_comms.cc)

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <time.h>
#include <string.h>

#include "mc_comms.hh"
#include "../constants.h"

FILE* uartInit(const char* device)
{
    static FILE *serial = fopen(device, "r+");

    /* Making fgets non-blocking
    http://stackoverflow.com/a/6055774 */
    int fd = fileno(serial);
    int flags = fcntl(fd, F_GETFL, 0);
    flags |= O_NONBLOCK;
    fcntl(fd, F_SETFL, flags);

    /* fclose(serial); */

    return serial;
}

void uartClose(FILE* uartDevice)
{
    fclose(uartDevice);
}

int uartSendRaw(char* string, FILE* uartDevice)
{
    if (strlen(string) > SENDBUFFER_SIZE)
    {
        return 1;
    }
    fwrite(string, sizeof(char), SENDBUFFER_SIZE, uartDevice);
    return 0;
}

int uartSendCommand(uint8_t command, int16_t data, FILE* uartDevice)
{
    char toSend[5];
    if ((command < 0x20) | (command > RECV_MAX))
    {
        return 1;
    }

    /*
        Convert 8 bit command and 16 bit data in to a 24 bit string.
        8 bit command, 2x 8 bit data.
        CCCCCCCC|DDDDDDDD|DDDDDDDD
    */
    toSend[0] = (char)command;
    toSend[1] = (char)(data >> 8);
    toSend[2] = (char)(data & 0x00FF);
    toSend[3] = (char)'\n';
    toSend[4] = (char)'\0';

    uartSendRaw(toSend, uartDevice);

    return 0;
}

int uartReadRaw(FILE* uartDevice, char* recvBuffer)
{
    if (fgets(recvBuffer, RECVBUFFER_SIZE, uartDevice) != NULL)
    {
        return 0;
    }
    return 1;
}
```

Listing 8: PS4 Controller to RPi UART test. (rpi/tests/test-ps4-uart.cc)

```
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//  http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
//
// Copyright Drew Noakes 2013-2016
//
// File first modified by Charlie Mason 25/02/2017

#include <unistd.h>

#include "joystick.hh"
#include "mc_comms.hh"
#include "../constants.h"

// Negative values correspond to the direction in variable names (positive for opposite direction)
int16_t servoButton = 0; //Turn on/off electromagnet
int16_t modeButton = 0; //Acts as mode switch between joystick and motion control methods
int16_t throttleUp = 0; // Axis 1
int16_t yawCCW = 0;    // Axis 0
int16_t pitchForward = 0; // Axis 5 (Joystick) Axis 13 (Motion)
int16_t rollLeft = 0;   // Axis 2 (Joystick) Axis 11 (Motion)

int main(int argc, char** argv)
{
  const char uartDevice[] = "/dev/serial0";
  FILE* serialDevice = uartInit(uartDevice);
  char toSend[SENDBUFFER_SIZE] = "";

// Create an instance of Joystick
  Joystick joystick("/dev/input/js0");

  // Ensure that it was found and that we can use it
  if (joystick.isFound())
  {
    printf("Joystick Connected.\n");
  }
  else
  {
    printf("Joystick not detected, exiting.\n");
    exit(1);
  }

  while (true)
  {
    // Restrict rate
    usleep(1000);

    // Attempt to sample an event from the joystick
    JoystickEvent event;
    if (joystick.sample(&event))
    {
      if (event.isButton())
      {
   if (event.number == 1)
     {
       modeButton = event.value;
       printf("Button %u is %s\n", event.number, modeButton == 0 ? "up" : "down");
            uartSendCommand(MODE_BUTTON, modeButton, serialDevice);

            if (event.value == 0)
       {
         pitchForward = 0;
         rollLeft = 0;
         printf("Pitch and Roll reset, now in JOYSTICK mode\n");
       }
     }
   else if (event.number == 0)
     {
       servoButton = event.value;
       printf("Button %u is %s\n", event.number, servoButton == 0 ? "up" : "down");
            uartSendCommand(SERVO_BUTTON, servoButton, serialDevice);
          }
      }
      else if (event.isAxis())
      {
   if (modeButton && (event.number > 5 || event.number == 1))
   {
     switch(event.number)
     {
       case 1 : throttleUp = event.value;
                     uartSendCommand(THROTTLE_UP, throttleUp, serialDevice);
           break;
       case 11: rollLeft = -event.value;
                     uartSendCommand(ROLL_LEFT, rollLeft, serialDevice);
           break;
       case 13: pitchForward = -event.value;
                     uartSendCommand(PITCH_FORWARD, pitchForward, serialDevice);
           break;
     }
   printf("MOTION Throttle: %6d, Roll: %6d, Pitch: %6d\n", throttleUp, rollLeft, pitchForward);
   }
   else if (modeButton == 0 && event.number <= 5)
   {
     switch(event.number)
     {
       case 0 : yawCCW = event.value;
                     uartSendCommand(YAW_CCW, yawCCW, serialDevice);
           break;
       case 1 : throttleUp = event.value;
                     uartSendCommand(THROTTLE_UP, throttleUp, serialDevice);
           break;
       case 2 : rollLeft = event.value;
                     uartSendCommand(ROLL_LEFT, rollLeft, serialDevice);
           break;
       case 5 : pitchForward = event.value;
                     uartSendCommand(PITCH_FORWARD, pitchForward, serialDevice);
           break;
     }
   printf("JOYSTICK Throttle: %6d, Yaw: %6d, Roll: %6d, Pitch: %6d\n", throttleUp, yawCCW, rollLeft, pitchForward);
   }
      }
    }
  }
}
```

Listing 9: Header file for IMU. (Arduino/MPU9250_reg.h)

```
#ifndef MPU9250_REG_H_INCLUDED
#define MPU9250_REG_H_INCLUDED

```

```c
#define MPU9250_I2C_CLOCK_SPEED             400000UL // I2C is 400KHz max
#define MPU9250_WRITE                    0x68 // This address is used by MPU9250 when ADC0 pin is logic low
#define MPU9250_READ                     0x69 // This address is used by MPU9250 when ADC0 pin is logic high

// Note, this is the reset value for all registers except
// - Register 107 (0x01) Power Management 1
// - Register 117 (0x71) WHO_AM_I
#define REG_RESET                    0x00

// From section 7.5 SPI Interface
// SPI read and write operations are completed in 16 or more clock cycles (two or more bytes). The
// first byte contains the SPI A ddress, and the following byte(s) contain(s) the SPI data. The first
// bit of the first byte contains the Read/Write bit and indicates the Read (1) or Write (0) operation.
// The following 7 bits contain the Register Address. In cases of multiple-byte Read/Writes, data is
// two or more bytes...
#define READ_MASK                    0x80

// Self Test, Gyro
#define SELF_TEST_X_GYRO             0x00
#define SELF_TEST_Y_GYRO             0x01
#define SELF_TEST_Z_GYRO             0x02



// Self Test, Accelerometer
#define SELF_TEST_X_ACCEL            0x0d
#define SELF_TEST_Y_ACCEL            0x0e
#define SELF_TEST_Z_ACCEL            0x0f

 // Gyro Offset
#define XG_OFFSET_H                  0x13
#define XG_OFFSET_L                  0x14
#define YG_OFFSET_H                  0x15
#define YG_OFFSET_L                  0x16
#define ZG_OFFSET_H                  0x17
#define ZG_OFFSET_L                  0x18


#define SMPLRT_DIV                   0x19

// Config
#define CONFIG                       0x1a
#define GYRO_CONFIG                  0x1b
#define ACCEL_CONFIG                 0x1c
#define ACCEL_CONFIG_2               0x1d
#define LP_ACCEL_ODR                 0x1e

#define WOM_THR                      0x1f

#define FIFO_EN                      0x23

// I2C
#define I2C_MST_CTRL                 0x24
#define I2C_SLV0_ADDR                0x25
#define I2C_SLV0_REG                 0x26
#define I2C_SLV0_CTRL                0x27

#define I2C_SLV1_ADDR                0x28
#define I2C_SLV1_REG                 0x29
#define I2C_SLV1_CTRL                0x2a

#define I2C_SLV2_ADDR                0x2b
#define I2C_SLV2_REG                 0x2c
#define I2C_SLV2_CTRL                0x2d

#define I2C_SLV3_ADDR                0x2e
```

```
#define I2C_SLV3_REG                 0x2f
#define I2C_SLV3_CTRL                0x30

#define I2C_SLV4_ADDR                0x31
#define I2C_SLV4_REG                 0x32
#define I2C_SLV4_DO                  0x33
#define I2C_SLV4_CTRL                0x34
#define I2C_SLV4_DI                  0x35

#define I2C_MST_STATUS               0x36

#define INT_PIN_CFG                  0x37
#define INT_ENABLE                   0x38

#define DMP_INT_STATUS               0x39 // Check DMP Interrupt, see 0x6d

#define INT_STATUS                   0x3a

// Accel XOUT
#define ACCEL_XOUT_H                 0x3b
#define ACCEL_XOUT_L                 0x3c
#define ACCEL_YOUT_H                 0x3d
#define ACCEL_YOUT_L                 0x3e
#define ACCEL_ZOUT_H                 0x3f
#define ACCEL_ZOUT_L                 0x40

// Temp.
#define TEMP_OUT_H                   0x41
#define TEMP_OUT_L                   0x42

// Gyro.
#define GYRO_XOUT_H                  0x43
#define GYRO_XOUT_L                  0x44
#define GYRO_YOUT_H                  0x45
#define GYRO_YOUT_L                  0x46
#define GYRO_ZOUT_H                  0x47
#define GYRO_ZOUT_L                  0x48

// Ext. Sensor data
#define EXT_SENS_DATA_00             0x49
#define EXT_SENS_DATA_01             0x4a
#define EXT_SENS_DATA_02             0x4b
#define EXT_SENS_DATA_03             0x4c
#define EXT_SENS_DATA_04             0x4d
#define EXT_SENS_DATA_05             0x4e
#define EXT_SENS_DATA_06             0x4f
#define EXT_SENS_DATA_07             0x50
#define EXT_SENS_DATA_08             0x51
#define EXT_SENS_DATA_09             0x52
#define EXT_SENS_DATA_10             0x53
#define EXT_SENS_DATA_11             0x54
#define EXT_SENS_DATA_12             0x55
#define EXT_SENS_DATA_13             0x56
#define EXT_SENS_DATA_14             0x57
#define EXT_SENS_DATA_15             0x58
#define EXT_SENS_DATA_16             0x59
#define EXT_SENS_DATA_17             0x5a
#define EXT_SENS_DATA_18             0x5b
#define EXT_SENS_DATA_19             0x5c
#define EXT_SENS_DATA_20             0x5d
#define EXT_SENS_DATA_21             0x5e
#define EXT_SENS_DATA_22             0x5f
#define EXT_SENS_DATA_23             0x60

// I2C slave
#define I2C_SLV0_DO                  0x63
#define I2C_SLV1_DO                  0x64
#define I2C_SLV2_DO                  0x65
#define I2C_SLV3_DO                  0x66

#define I2C_MST_DELAY_CTRL           0x67

```

```

// Signal path
#define SIGNAL_PATH_RESET            0x68

// Motion detect
#define MOT_DETECT_CTRL              0x69

// User
#define USER_CTRL                    0x6a // Bit 7 enable DMP, bit 3 reset DMP. See 0x6d

// Power management
#define PWR_MGMT_1                   0x6b
#define PWR_MGMT_2                   0x6c

// ...Looked for notes on DMP features, but Invensense docs were lacking.
// Found kriswiner's Arduino sketch for Basic AHRS, and found values/notes for
// Digital Motion Processing registers.
//
// See https://github.com/kriswiner/MPU-9250/blob/master/MPU9250BasicAHRS.ino
#define DMP_BANK                     0x6d
#define DMP_RW_PNT                   0x6e
#define DMP_REG                      0x6f
#define DMP_REG_1                    0x70
#define DMP_REG_2                    0x71

// FIFO Count
#define FIFO_COUNTH                  0x72
#define FIFO_COUNTL                  0x73
#define FIFO_R_W                     0x74
#define WHO_AM_I                     0x75 //should return something else???

// Accel. offset
#define XA_OFFSET_H                  0x77
#define XA_OFFSET_L                  0x78
#define YA_OFFSET_H                  0x7a
#define YA_OFFSET_L                  0x7b
#define ZA_OFFSET_H                  0x7d
#define ZA_OFFSET_L                  0x7e

#endif
```

Listing 10: I2C header file. (Arduino/i2c.h)

```
#ifndef I2C_MASTER_H
#define I2C_MASTER_H

#define I2C_READ 0x01
#define I2C_WRITE 0x00

#include <avr/io.h>



void i2c_init(void);
uint8_t i2c_start(uint8_t address);
uint8_t i2c_write(uint8_t data);
uint8_t i2c_read_ack(void);
uint8_t i2c_read_nack(void);
uint8_t i2c_transmit(uint8_t address, uint8_t* data, uint16_t length);
uint8_t i2c_receive(uint8_t address, uint8_t* data, uint16_t length);
uint8_t i2c_writeReg(uint8_t devaddr, uint8_t regaddr, uint8_t* data, uint16_t length);
uint8_t i2c_readReg(uint8_t devaddr, uint8_t regaddr, uint8_t* data, uint16_t length);
void i2c_stop(void);

#endif // I2C_MASTER_H
```

Listing 11: I2C library. (Arduino/i2c.c)

```
#ifndef F_CPU
#define F_CPU 16000000UL
#endif

#include <avr/io.h>
#include <util/twi.h>

#include "i2c.h"

#define F_SCL 100000UL // SCL frequency
#define Prescaler 1
#define TWBR_val ((((F_CPU / F_SCL) / Prescaler) - 16 ) / 2)

void i2c_init(void)
{
   //TWBR = (uint8_t)TWBR_val;
   TWSR = 0x00;
    TWBR = 0x0C;
    //enable TWI
    TWCR = (1<<TWEN);
}

uint8_t i2c_start(uint8_t address)
{
   // reset TWI control register
   TWCR = 0;
   // transmit START condition
   TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
   // wait for end of transmission
   while( !(TWCR & (1<<TWINT)) );

   // check if the start condition was successfully transmitted
   if((TWSR & 0xF8) != TW_START){ return 1; }

   // load slave address into data register
   TWDR = address;
   // start transmission of address
   TWCR = (1<<TWINT) | (1<<TWEN);
   // wait for end of transmission
   while( !(TWCR & (1<<TWINT)) );

   // check if the device has acknowledged the READ / WRITE mode
   uint8_t twst = TW_STATUS & 0xF8;
   if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) ) return 1;

   return 0;
}

uint8_t i2c_write(uint8_t data)
{
   // load data into data register
   TWDR = data;
   // start transmission of data
   TWCR = (1<<TWINT) | (1<<TWEN);
   // wait for end of transmission
   while( !(TWCR & (1<<TWINT)) );

   if( (TWSR & 0xF8) != TW_MT_DATA_ACK ){ return 1; }

   return 0;
}

uint8_t i2c_read_ack(void)
{

   // start TWI module and acknowledge data after reception
   TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
   // wait for end of transmission
   while((TWCR & (1<<TWINT)) ==0);
   // return received data from TWDR
```

```c
   return TWDR;
}

uint8_t i2c_read_nack(void)
{

   // start receiving without acknowledging reception
   TWCR = (1<<TWINT) | (1<<TWEN);
   // wait for end of transmission
   while( !(TWCR & (1<<TWINT)) );
   // return received data from TWDR
   return TWDR;
}

uint8_t i2c_transmit(uint8_t address, uint8_t* data, uint16_t length)
{
   if (i2c_start(address | I2C_WRITE)) return 1;
   uint16_t i;
   for (i = 0; i < length; i++)
   {
      if (i2c_write(data[i])) return 1;
   }

   i2c_stop();

   return 0;
}

uint8_t i2c_receive(uint8_t address, uint8_t* data, uint16_t length)
{
   if (i2c_start(address | I2C_READ)) return 1;
   uint16_t i;
   for (i = 0; i < (length-1); i++)
   {
      data[i] = i2c_read_ack();
   }
   data[(length-1)] = i2c_read_nack();

   i2c_stop();

   return 0;
}

uint8_t i2c_writeReg(uint8_t devaddr, uint8_t regaddr, uint8_t* data, uint16_t length)
{
   if (i2c_start(devaddr | 0x00)) return 1;

   i2c_write(regaddr);
   uint16_t i;
   for (i = 0; i < length; i++)
   {
      if (i2c_write(data[i])) return 1;
   }

   i2c_stop();

   return 0;
}

uint8_t i2c_readReg(uint8_t devaddr, uint8_t regaddr, uint8_t* data, uint16_t length)
{
   if (i2c_start(devaddr)) return 1;

   i2c_write(regaddr);

   if (i2c_start(devaddr | 0x01)) return 1;

   uint16_t i;
   for (i = 0; i < (length-1); i++)
   {
      data[i] = i2c_read_ack();
   }
   data[(length-1)] = i2c_read_nack();

   i2c_stop();

   return 0;
}

void i2c_stop(void)
{
   // transmit STOP condition
   TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
}
```

Listing 12: IR Sensor header file. (Arduino/ir.h)

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>


#define ADC_PIN 0
#define ADC_THRESHOLD 512

uint16_t adc_read(uint8_t adcx);
void adc_init(void);
```

Listing 13: IR Sensor library. (Arduino/ir.c)

```
#include "ir.h"

void adc_init(void)
{
 // Select Vref=AVcc
 ADMUX |= (1<<REFS0);
 //set prescaller to 128 and enable ADC
 ADCSRA |= (1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)|(1<<ADEN);
}



uint16_t adc_read(uint8_t adcx) {
  /* adcx is the analog pin we want to use. ADMUX's first few bits are
   * the binary representations of the numbers of the pins so we can
   * just 'OR' the pin's number with ADMUX to select that pin.
   * We first zero the four bits by setting ADMUX equal to its higher
   * four bits. */
  ADMUX &= 0xf0;
  ADMUX |= adcx;

  /* This starts the conversion. */
  ADCSRA |= _BV(ADSC);

  /* This is an idle loop that just wait around until the conversion
   * is finished. It constantly checks ADCSRA's ADSC bit, which we just
   * set above, to see if it is still set. This bit is automatically
   * reset (zeroed) when the conversion is ready so if we do this in
   * a loop the loop will just go until the conversion is ready. */
  while ( (ADCSRA & _BV(ADSC)) );

  /* Finally, we return the converted value to the calling function. */
  return ADC;
}
```

Listing 14: PID Control header file. (Arduino/pid.h)

```
#include <util/delay.h>

#ifndef PID_H_
```

```
#define PID_H_

extern double Kp_roll,  Ki_roll,  Kd_roll;
extern double Kp_pitch, Ki_pitch,  Kd_pitch;
extern double Kp_yall,  Ki_yall,  Kd_yall;

int PID(double measured_angle, double desired_angle, double dt, double Kp, double Ki, double Kd); // PID control function

#endif
```

Listing 15: PID Control library. (Arduino/pid.c)

```
#include "pid.h"

double Kp_roll = 70;
double Ki_roll = 10;
double Kd_roll = 15;

double Kp_pitch = 70;
double Ki_pitch = 10;
double Kd_pitch = 15;

double Kp_yall = 70;
double Ki_yall = 10;
double Kd_yall = 15;

double integral = 0;
double previous_error = 0;

int PID(double measured_angle, double desired_angle, double dt, double Kp, double Ki, double Kd)
{

double output;
double error = desired_angle - measured_angle;
integral += error*dt;
double derivative = (error-previous_error)/dt;

   if ((integral < -0.01)||(integral > 0.01))
      integral = 0;                   // prevent integral wind-up

output = Kp*error + Ki*integral + Kd*derivative; //  calculate new value
previous_error = error;
_delay_ms(dt);

return(output);

}
```

Listing 16: Ultrasonic Sensor header file. (Arduino/sonar.h)

```
/*!
 * ***********************************************************************************************
 * \file sonar.h
 * \brief Interfacing HC-SR04 Ultrasonic Sensor Module (Sonar)
 *
 * \author      :  Praveen Kumar
 * \date        :  Mar 24, 2014
 * Copyright(c) :  Praveen Kumar - www.veerobot.com
 * Description  :  Interfacing HC-SR04 Ultrasonic Sensor Module (Sonar). Program is tested on
 *                 Draco - AVR Development board available at www.veerobot.com/store which has an
 *                 ATmega328P microcontroller. If you replace that with any other 28 pin AVR
 *                 microcontroller, be sure to modify registers accordingly.
 *
 * LICENSE      :  Redistribution and use in source and/or binary forms, with or
```

```
    without modification,
 * are permitted provided that the following conditions are met:
 *
 * - Redistributions of source code must retain this copyright notice, list of conditions and disclaimer.
 * - Redistributions in binary form must reproduce this copyright notice, list of conditions and disclaimer in
 *     documentation and/or other materials provided with the distribution.
 *
 * DISCLAIMER  :   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" WITHOUT ANY
 * KIND OF WARRANTIES. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE
 *
 * ***********************************************************************************************
 */
#ifndef SONAR_H_
#define SONAR_H_

#ifndef F_CPU
    #define F_CPU 1000000UL  // CPU Frequency
#endif

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

/*...- . . .-. --- -... --- -
 * Define Ports and Pins as required
 * Modify Maximum response time and delay as required
 * MAX_RESP_TIME : default: 300
 * DELAY_BETWEEN_TESTS : default: 50
 */
#define TRIG_DDR  DDRD          // Trigger Port
#define TRIG_PORT PORTD
#define TRIG_PIN  PIND
#define TRIG_BIT  PD2           // Trigger Pin

#define ECHO_DDR  DDRD          // Echo Port
#define ECHO_PORT PORTD
#define ECHO_PIN  PIND
#define ECHO_BIT  PD3           // Echo Pin

// Speed of sound
// Default: 343 meters per second in dry air at room temperature (~20C)
#define SPEED_OF_SOUND 343
#define MAX_SONAR_RANGE 10      // This is trigger + echo range (in meters) for SR04
#define DELAY_BETWEEN_TESTS 500 // Echo canceling time between sampling. Default: 500us
#define TIMER_MAX 65535         // 65535 for 16 bit timer and 255 for 8 bit timer

/* ...- . . .-. --- -... --- -
 * Do not change anything further unless you know what you're doing
 * */
#define TRIG_ERROR -1
#define ECHO_ERROR -2

#define CYCLES_PER_US (F_CPU/1000000)// instructions per microsecond
#define CYCLES_PER_MS (F_CPU/1000)  // instructions per millisecond
// Timeout. Decreasing this decreases measuring distance
// but gives faster sampling
#define SONAR_TIMEOUT ((F_CPU*MAX_SONAR_RANGE)/SPEED_OF_SOUND)

#define TRIG_INPUT_MODE() TRIG_DDR &= ~(1<<TRIG_BIT)
#define TRIG_OUTPUT_MODE() TRIG_DDR |= (1<<TRIG_BIT)
#define TRIG_LOW() TRIG_PORT &= ~(1<<TRIG_BIT)
#define TRIG_HIGH() TRIG_PORT |=(1<<TRIG_BIT)

#define ECHO_INPUT_MODE() ECHO_DDR &= ~(1<<ECHO_BIT)
```

```
#define ECHO_OUTPUT_MODE() ECHO_DDR |= (1<<ECHO_BIT)
#define ECHO_LOW() ECHO_PORT &= ~(1<<ECHO_BIT)
#define ECHO_HIGH() ECHO_PORT |=(1<<ECHO_BIT)

#define CONVERT_TO_CM ((10000*2)/SPEED_OF_SOUND) // or simply 58

/** ...- . . .-. --- -... --- -
 * @brief  Initiate Ports for Trigger and Echo pins
 * @param  void
 * @return none
*/
void init_sonar();

/**  ...- . . .-. --- -... --- -
 * @brief  Send 10us pulse on Ultrasonic Trigger pin
 * @param  void
 * @return none
*/
void trigger_sonar();

/**  ...- . . .-. --- -... --- -
 * @brief  Calculate and store echo time and return distance
 * @param  void
 * @return unsigned int
 * Usage   int foo = read_sonar();
*/
unsigned int read_sonar();

#endif /* SONAR_H_ */
```

Listing 17: Ultrasonic Sensor library. (Arduino/sonar.c)

```
/*!
 * **********************************************************************************************
 * \file sonar.c
 * \brief Interfacing HC-SR04 Ultrasonic Sensor Module (Sonar)
 *
 * \author     :   Praveen Kumar
 * \date       :   Mar 24, 2014
 * Copyright(c)       :   Praveen Kumar - www.veerobot.com
 * Description        :   refer sonar.h
 *
 * LICENSE     :   Refer sonar.h
 *
 * **********************************************************************************************
 */

#include "sonar.h"

volatile uint32_t overFlowCounter = 0;
volatile uint32_t trig_counter = 0;
volatile uint32_t no_of_ticks = 0;

/********** ...- . . .-. --- -... --- - *********************************
 * Initiate Ultrasonic Module Ports and Pins
 * Input:  none
 * Returns: none
*********** ...- . . .-. --- -... --- - *********************************/
void init_sonar(){
    TRIG_OUTPUT_MODE();   // Set Trigger pin as output
    ECHO_INPUT_MODE();    // Set Echo pin as input
}

/********** ...- . . .-. --- -... --- - *********************************
 * Send 10us pulse on Sonar Trigger pin
 * 1.  Clear trigger pin before sending a pulse
 * 2.  Send high pulse to trigger pin for 10us
 * 3.  Clear trigger pin to pull it trigger pin low
 *  Input:   none
 *  Returns: none
********** ...- . . .-. --- -... --- - *********************************/
void trigger_sonar(){
    TRIG_LOW();          // Clear pin before setting it high
    _delay_us(1);        // Clear to zero and give time for electronics to set
    TRIG_HIGH();         // Set pin high
    _delay_us(12);       // Send high pulse for minimum 10us
    TRIG_LOW();          // Clear pin
    _delay_us(1);        // Delay not required, but just in case...
}

/********** ...- . . .-. --- -... --- - *********************************
 * Increment timer on each overflow
 * Input:  none
 * Returns: none
********** ...- . . .-. --- -... --- - *********************************/
ISR(TIMER1_OVF_vect){ // Timer1 overflow interrupt
    overFlowCounter++;
    TCNT1=0;
}

/********** ...- . . .-. --- -... --- - *********************************
 * Calculate and store echo time and return distance
 * Input:  none
 * Returns: 1. -1     :   Indicates trigger error. Could not pull trigger high
 *         2. -2      :   Indicates echo error. No echo received within range
 *         3. Distance :  Sonar calculated distance in cm.
********** ...- . . .-. --- -... --- - *********************************/
unsigned int read_sonar(){
    int dist_in_cm = 0;
    init_sonar();                       // Setup pins and ports
    trigger_sonar();                    // send a 10us high pulse

    while(!(ECHO_PIN & (1<<ECHO_BIT))){ // while echo pin is still low
        trig_counter++;
         uint32_t max_response_time = SONAR_TIMEOUT;
        if (trig_counter > max_response_time){ // SONAR_TIMEOUT
            return TRIG_ERROR;
        }
    }

    TCNT1=0;                            // reset timer
    TCCR1B |= (1<<CS10);              // start 16 bit timer with no prescaler
    TIMSK1 |= (1<<TOIE1);             // enable overflow interrupt on timer1
    overFlowCounter=0;                  // reset overflow counter
    sei();                              // enable global interrupts

    while((ECHO_PIN & (1<<ECHO_BIT))){ // while echo pin is still high
        if (((overFlowCounter*TIMER_MAX)+TCNT1) > SONAR_TIMEOUT){
            return ECHO_ERROR;          // No echo within sonar range
        }
    };

    TCCR1B = 0x00;                      // stop 16 bit timer with no prescaler
    cli();                              // disable global interrupts
    no_of_ticks = ((overFlowCounter*TIMER_MAX)+TCNT1); // counter count
    dist_in_cm = (no_of_ticks/(CONVERT_TO_CM*CYCLES_PER_US)); // distance in cm
    return (dist_in_cm );
}
```

Listing 18: UART header file. (Arduino/uart.h)

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#define BLINK_DELAY_MS 1000
#define BAUDRATE 9600
#define BAUD_PRESCALLER (((F_CPU / (BAUDRATE * 16UL))) - 1)

```

```

void USART_init(void);
unsigned char USART_receive(void);
void USART_send( unsigned char data);
void USART_putstring(char* StringPtr);
```

Listing 19: UART library. (Arduino/uart.c)

```
#include "uart.h"

void USART_init(void){

 UBRR0H = (uint8_t)(BAUD_PRESCALLER>>8);
 UBRR0L = (uint8_t)(BAUD_PRESCALLER);
 UCSR0B = (1<<RXEN0)|(1<<TXEN0);
 UCSR0C = (3<<UCSZ00);
}

unsigned char USART_receive(void){

 while(!(UCSR0A & (1<<RXC0)));
 return UDR0;
 //i = atoi (String);

}

void USART_send( unsigned char data){

 while(!(UCSR0A & (1<<UDRE0)));
 UDR0 = data;

}

void USART_putstring(char* StringPtr){

while(*StringPtr != 0x00){
 USART_send(*StringPtr);
 StringPtr++;}

}
```

Listing 20: Seeeduino main. (Arduino/main.c)

```
//PD2 Trigger
//PD3 Echo
//PC0 ACD IR sensor

#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include <stdlib.h>
#include "sonar.h"
#include "uart.h"
#include "ir.h"
#include "i2c.h"
#include "MPU9250_reg.h"
#include <math.h>

void init_MPU9250(void);
float getacc(void);
int16_t whoami(void);

char buffer[8];
int16_t distance;
int16_t raw_x = 0;
int16_t raw_y = 0;
int16_t raw_z = 0;
int16_t value;


```

```

enum Ascale {
  AFS_2G = 0,
  AFS_4G,
  AFS_8G,
  AFS_16G
};

enum Gscale {
  GFS_250DPS = 0,
  GFS_500DPS,
  GFS_1000DPS,
  GFS_2000DPS
};
uint8_t Ascale = AFS_2G;
uint8_t Gscale = GFS_250DPS;

int main (void)
{
  DDRB |= _BV(DDB5); /* set pin 5 of PORTB for output*/
  USART_init();
  i2c_init();
  //USART_putstring("initialised i2c");

  init_MPU9250();

//USART_putstring("initialised imu \t");
  while(1) {

  // acc = getacc();


  whoami();
//USART_putstring("after whoami function \t");
  //itoa (value,buffer,10);
  //USART_putstring(buffer);
  //USART_putstring("\n");





    _delay_ms(1000);
 }

 return 0;
}


void init_MPU9250(void){

 // USART_putstring("start int loop");
  uint8_t regv = 0x01;

  itoa (regv,buffer,16);
  USART_putstring(buffer);

  uint8_t c;
 // i2c_start(MPU9250_WRITE);
 // i2c_write(PWR_MGMT_1);
//  i2c_write(0x00);

//i2c_writeReg(MPU9250_WRITE, PWR_MGMT_1, 0x00, 8);

c=0x00;
i2c_writeReg(MPU9250_WRITE, PWR_MGMT_1, &c, 8);
//regv= i2c_readReg(MPU9250_WRITE, PWR_MGMT_1, &regv, 8);

  i2c_start(MPU9250_WRITE);
  i2c_write(PWR_MGMT_1);
  i2c_stop();

  i2c_start(MPU9250_WRITE);
  regv = ((uint8_t)i2c_read_ack()<<8);
  regv |= i2c_read_ack();
  i2c_stop();

  itoa (regv,buffer,16);
  USART_putstring(buffer);
}


int16_t whoami(void){
//i2c_readReg(MPU9250_WRITE, WHO_AM_I, value, 8);

//USART_putstring("entered whoami\t");
  i2c_start(MPU9250_WRITE);
  i2c_write(WHO_AM_I); // set pointer to X axis MSB
  i2c_stop();
//USART_putstring("imu write function \t");

  i2c_start(MPU9250_WRITE);
//USART_putstring("1 \t");
  value = ((uint8_t)i2c_read_ack());


 // USART_putstring("2 \n");
  //value |= i2c_read_ack();
i2c_stop();

//USART_putstring("after read \n");


//  itoa (value,buffer,16);
 // USART_putstring(buffer);
 // USART_putstring("\n");
//USART_putstring("after loop\n");

return value;
}
```