Crane Positional Sensor

Group: sdmay21-20

Problem Statement

Stellar Industries needs a rotational sensor to calculate the angle at which their crane is positioned according to their truck base. The sensor is needed to provide that information to their operators so they can uphold their safety standards.

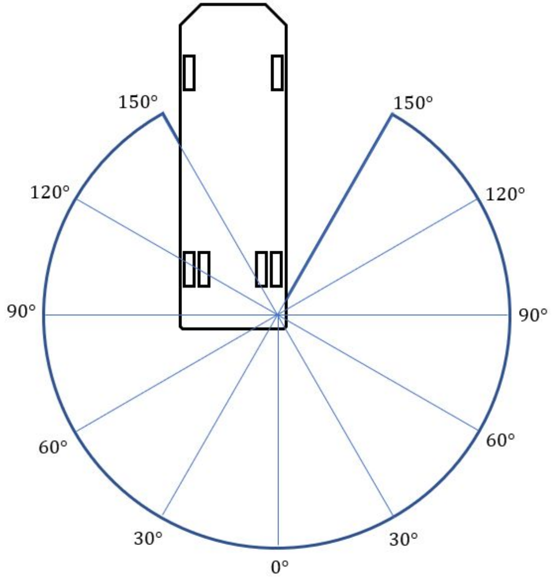


Figure 1: Stability Diagram

Functional Requirements

- Compute the angle of rotation, with a maximum error of ±5°
- The sensor and MCU must operate accurately between -40°F and 160°F
- The sensor and sensor housing must be mountable to the outside of the crane
- The sensor must operate accurately during inclement weather including rain, snow and humid conditions
- Must be powered by a 12V DC power supply
- Communicate with the controller via CAN bus protocol
- Must be able to update the user about the angle of rotation in real time

Non-Functional Requirements

- The solution must be cost-efficient
- The solution must be able to be reproduced for installation on multiple cranes

Technical/Other Constraints/Considerations

Technical constraints include:

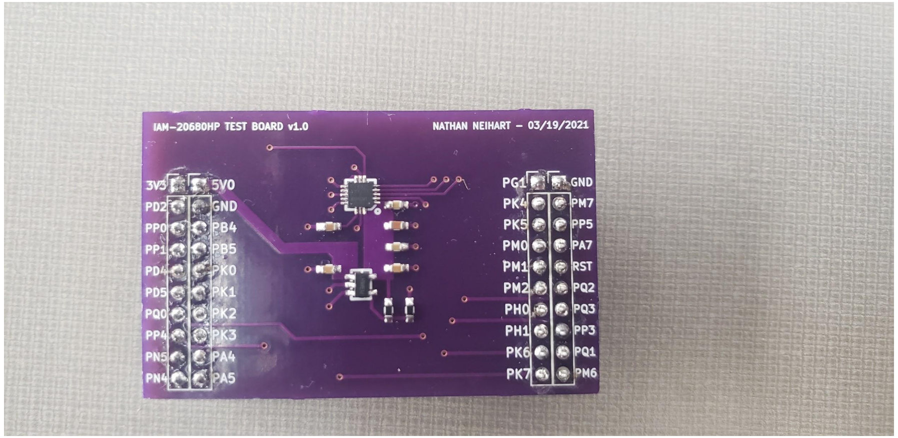
- The design must not use a mechanical gear for the rotational sensor

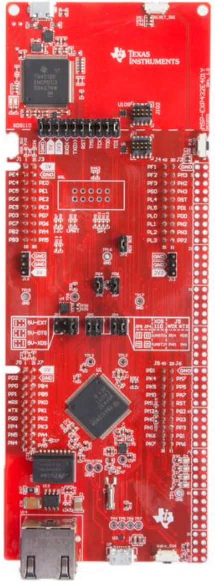
Technical considerations include:

- The sensor design should be easily mountable/attachable to the crane to avoid having to take any parts of the crane off

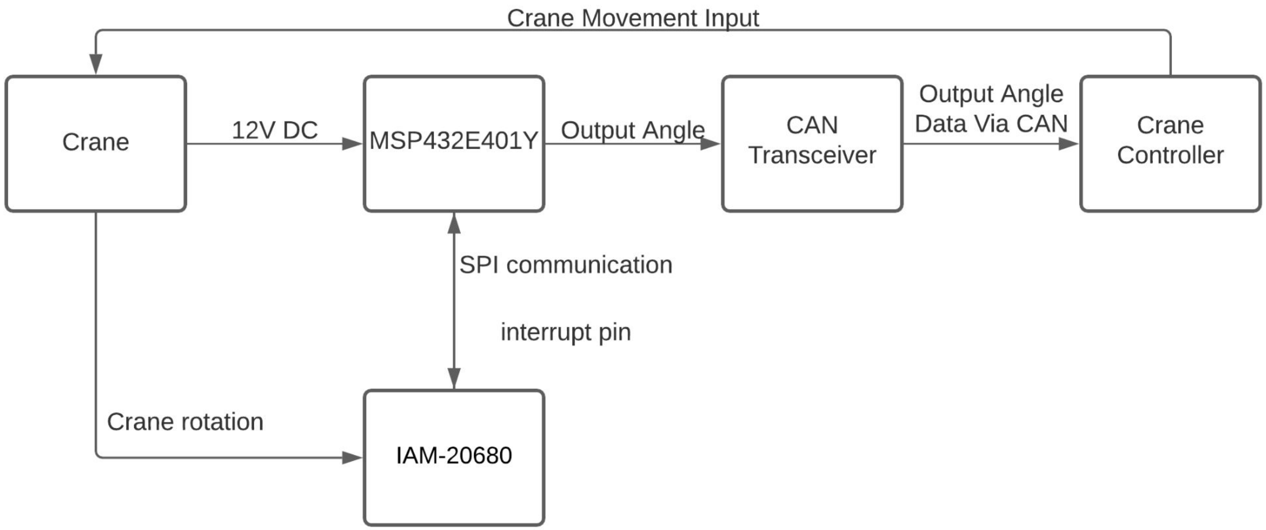
Components

- MSP432E401Y
- TDK IAM-20680HP

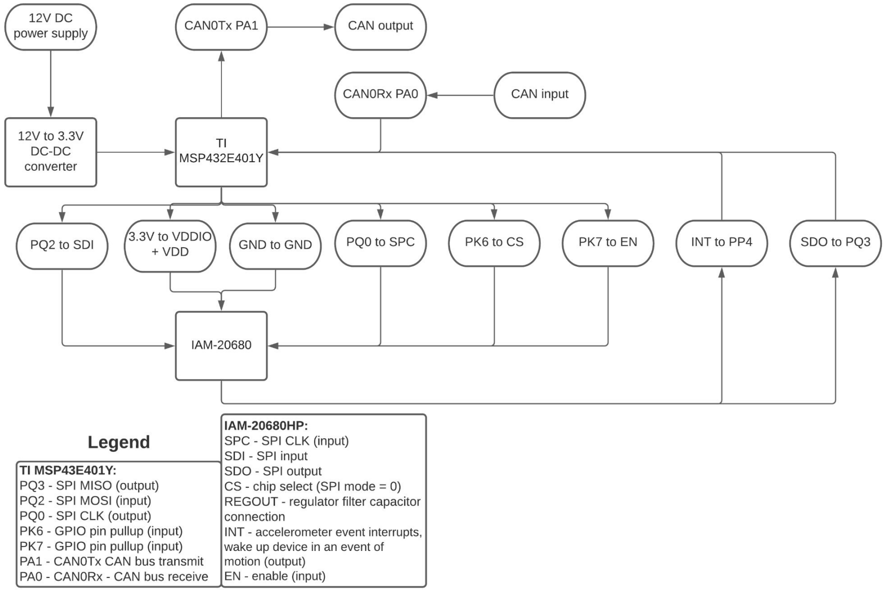




Block Diagram



Component Block Diagram



Calculations

- Calculate average offset
- Apply high-pass filter
- Initialize angle to 180°
- Read data from gyro sensor (velocity)
- Scale to degrees/second
- Integrate over time
- Format to (180° to -179°)

```
uint8_t initial_angle = 180;
uint8_t angle = initial_angle;
```

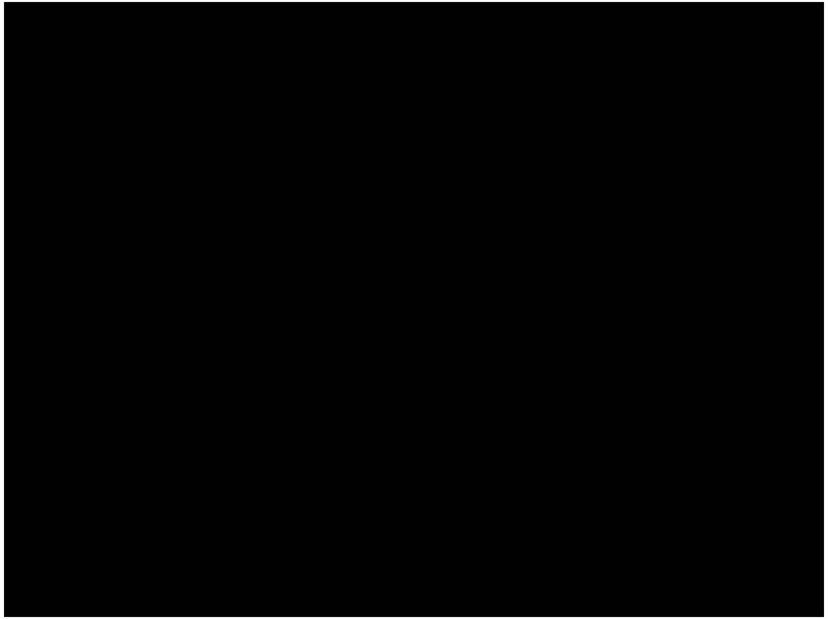
```
    angle = angle + dps_gyro_X * .12;
    if (angle > 180){
        angle -= 360;
    }
    else if (angle < -179){
        angle += 360;
    }
    Display_printf(display, 0, 0, "Angle %2.2", angle);
    // need to exit loop somehow, probably if the machine is b
}
```

Calculation Changes

$$\theta = tan^{-1}(\frac{ycos\alpha - zsin\alpha}{xcos\beta + zsin\beta})$$

- Previous method involved 3 accelerometers, 2 gyros
- α and ß are X- and Y-rotation
- Highly accurate in theory
    - Accel data is inaccurate (short-term)
    - Computationally expensive
    - Non-rotating sensor

Calculations



SPI

- Configuring Registers
  - TDK-20680HP Data sheet
- Configuring SPI parameters
  - TDK-20680HP Data sheet
  - MSP432E401Y Data sheet
- Transmit and Receive
  - Code Composer Studio TI examples

Configuring Registers

- 36 overall Registers
  - Configure registers
  - Disable/Default registers
  - Read Registers

```
///////////////// REGISTERS TO CONFIGURE TDK /////////////////////////
#define PWR_MGMT_1_ADD      0x6B      // Address Write
#define PWR_MGMT_1_ADD_READ 0xEB      // Address Read
#define WHO_AM_I            0xF5
#define PWR_MGMT_1_DAT      0x81      // Data
#define PWR_MGMT_2_ADD      0x6C      // Address
#define PWR_MGMT_2_DAT      0x00        // Data
#define GYRO_CONFIG_ADD     0x1B      // Address
#define GYRO_CONFIG_DAT     0x08      // Data
#define ACCEL_CONFIG_ADD    0x1C      // Address
#define ACCEL_CONFIG_DAT    0x08      // Data
#define ACCEL_CONFIG2_ADD   0x1D      // Address
#define ACCEL_CONFIG2_DAT   0x15      // Data

///////////////// REGISTERS TO BE SET OFF OR TO DEFAULT /////////////
#define FIFO_EN_ADD         0x23      // Address
#define FIFO_EN_DAT         0x00      // Data
#define FSYNC_INT_ADD       0x36      // Address
#define FSYNC_INT_DAT       0x00      // Data
#define INT_PIN_CFG_ADD     0x37      // Address
#define INT_PIN_CFG_DAT     0x00      // Data
#define INT_ENABLE_ADD      0x38      // Address
#define INT_ENABLE_DAT      0x00      // Data
#define INT_STATUS_ADD      0x3A      // Address
#define INT_STATUS_DAT      0x00      // Data

///////////////// REGISTERS TO TAKE VALUES FROM /////////////////////
#define ACCEL_XOUT_H        0xBB
#define ACCEL_XOUT_L        0xBC
#define ACCEL_YOUT_H        0xBD
#define ACCEL_YOUT_L        0xBE
#define ACCEL_ZOUT_H        0xBF
#define ACCEL_ZOUT_L        0xC0
#define GYRO_XOUT_H         0xC3
#define GYRO_XOUT_L         0xC4
#define GYRO_YOUT_H         0xC5
#define GYRO_YOUT_L         0xC6
#define GYRO_ZOUT_H         0xC7
#define GYRO_ZOUT_L         0xC8
```

Configuring SPI Parameters

- Setting parameter values

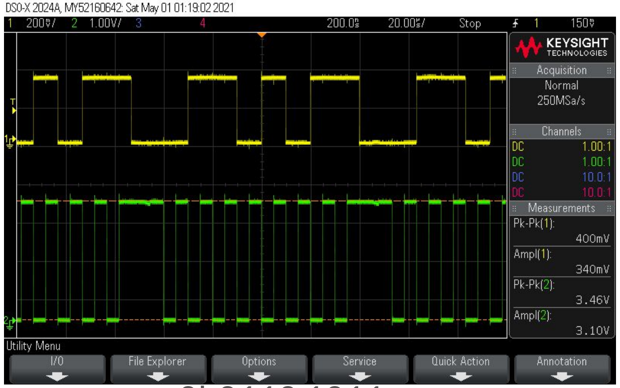
```
SPI_Handle spi;
SPI_Params spiParams;
SPI_Transaction t0;

SPI_Params_init(&spiParams);
spiParams.transferMode = SPI_MODE_BLOCKING;
spiParams.mode = SPI_MASTER;
spiParams.frameFormat = SPI_POL1_PHA1;
spiParams.bitRate = 100000;
spiParams.dataSize = 8;

spi = SPI_open(CONFIG_SPI_MASTER, &spiParams);
```

Transmitting and Receiving

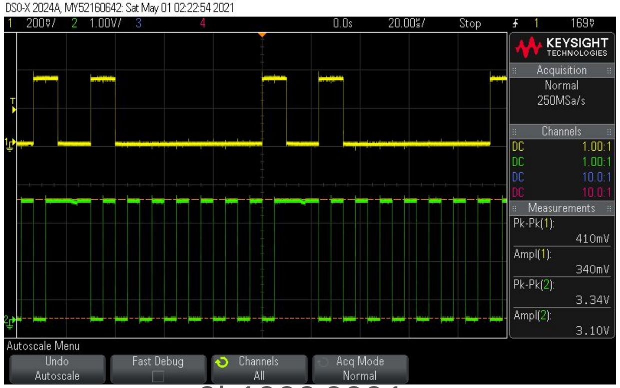
- Transaction
  - count
  - txBuf
  - rxBuf

```
t0.count = 1;
t0.txBuf = (void*) &transmitBuffer;
t0.rxBuf = (void*) &receiveBuffer;

transmitBuffer = transmit[0];
SPI_transfer(spi, &t0);                    // send write address

transmitBuffer = transmit[1];
SPI_transfer(spi, &t0);                    // send data
```

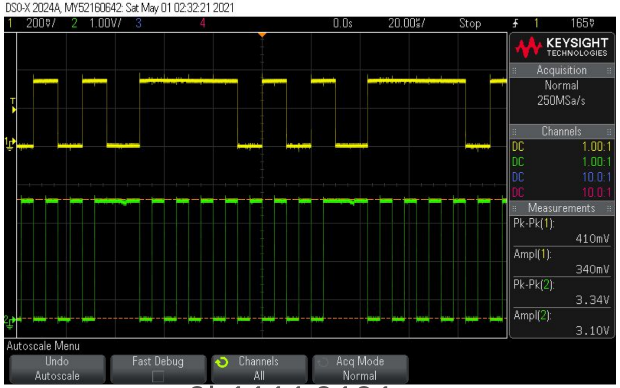
Oscilloscope Traces



0b0110 1011



0b1000 0001



0b1111 0101

SPI Complications

- Receive buffer

Expression	Type	Value	Address
> transmit	unsigned char[23]	[107 'k',129 '\x81',245 '\xf5',0 '\x00',0 '\x00'...]	0x200212B8
transmitBuffer	unsigned char	245 '\xf5'	0x200212F5
receiveBuffer	unsigned char	0 '\x00'	0x200212F4

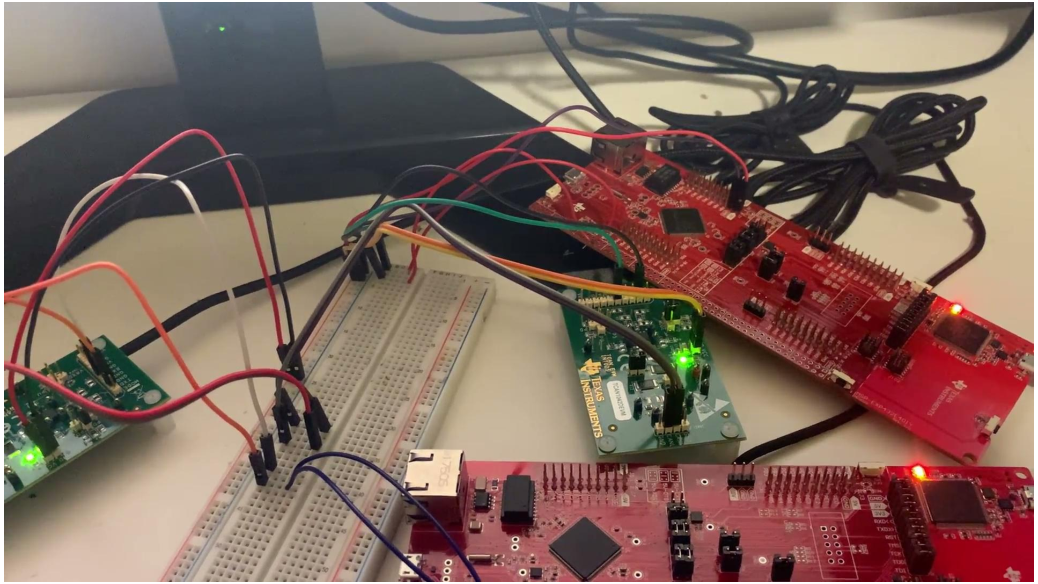
CAN bus

- GPIO Interrupt handler
  - Set pin PP4 as an interrupt
- Configure CAN and UART
  - UART is being used to display messages for testing purposes
- SYS_CLOCK & CAN_ADRESS set as macros as they may change depending on on the system
- Main function will send and print the message in UART along with displaying any error messages
- Pins used for CAN bus include PA0, PA1 and PP4
  - PA0 and PA1 are receive and transmit respectively
  - PP4 is a GPIO pin configured as an interrupt connected to the interrupt pin on the IAM-60860HP and will trigger txMsg to true to allow for messages to be transmitted.

CAN bus Code

```
/* A new message will be sent after pp1 is activated */
if (txMsg)
{
    /* Print a message to the console showing the message count and the
     * contents of the message being sent, this message is used for testing*/
    //test
    UARTprintf("Sending msg 0x%03X: 0x%02X 0x%02X 0x%02X 0x%02X 0x%02X 0x%02X 0x%02X 0x%02X 0x%02X 0x%02X 0x%02X 0x%02X 0x%02X 0x%02X 0x%02X 0x%02X",
               sCANMessage.ui32MsgID, msgData[0], msgData[1], msgData[2],
               msgData[3], msgData[4], msgData[5], msgData[6], msgData[7], msgData[8], msgData[9], msgData[10], msgData[11], msgData[12], msgData[13], msgData[14], msgData[15]);

    /* Send the CAN message using object number 1 */
    MAP_CANMessageSet(CAN0_BASE, 1, &sCANMessage, MSG_OBJ_TYPE_TX);

    /* Check the error flag to see if errors occurred */
    if(errFlag)
    {
        UARTprintf(" error\n");
    }
    else
    {
        /* If no errors then print "message sent" */
        UARTprintf(" message sent\n");
    }
```

CAN bus Demo



Thank You

Any Questions?

Advisor: Nathan Neihart

Client: Stellar Industries & Nathan Meyer

Team members: Andrew Jacobson, Eli Davidson, Wyatt Syhlman & Nikhil Sharma

Email: sdmay21-20@iastate.edu