Crane Positional Sensor

FINAL REPORT

Team 20
Stellar Industries - Client
Eli Davidson
Andrew Jacobson
Nikhil Sharma
Wyatt Syhlman
Adviser: Nathan Neihart
sdmay21-20@iastate.edu
Team Website: https://sdmay21-20.sd.ece.iastate.edu/

Revised: 4/25/2021

Table of Contents

1. Introduction

1.1 Acknowledgement

We would like to thank Nathan Neihart for all of the technical help and instruction we received during our weekly meetings.

1.2 Problem Statement

Stellar Industries currently does not have the appropriate sensors to communicate the degrees of rotation to their crane operators. The main goal is to provide Stellar's crane operators with the proper angle at which the crane is rotating around the truck, while it is in operation, to allow for the safe use of the crane. The reason they need this sensor is that the truck is more stable depending on the angle of the crane while moving a load. Figure 1 shows the Stability Capacity Chart similar to what Stellar currently uses to test different truck's percentage of rated capacity for these angles. That means that Stellar tests at what angles the truck can hold a maximum load and at what angles the operator has to be aware of, for the truck being able to tip over, if the weight of the object is too much or if the object is too far away from the truck. Stellar is currently using the judgment of their crane operators to determine the angle at which the crane is positioned according to the point of origin. However, they wish to make this more absolute, for additional safety in their practice. In Figure 1, the point of origin is the 0° mark. The sensor also must be able to operate in different elements of the environment.

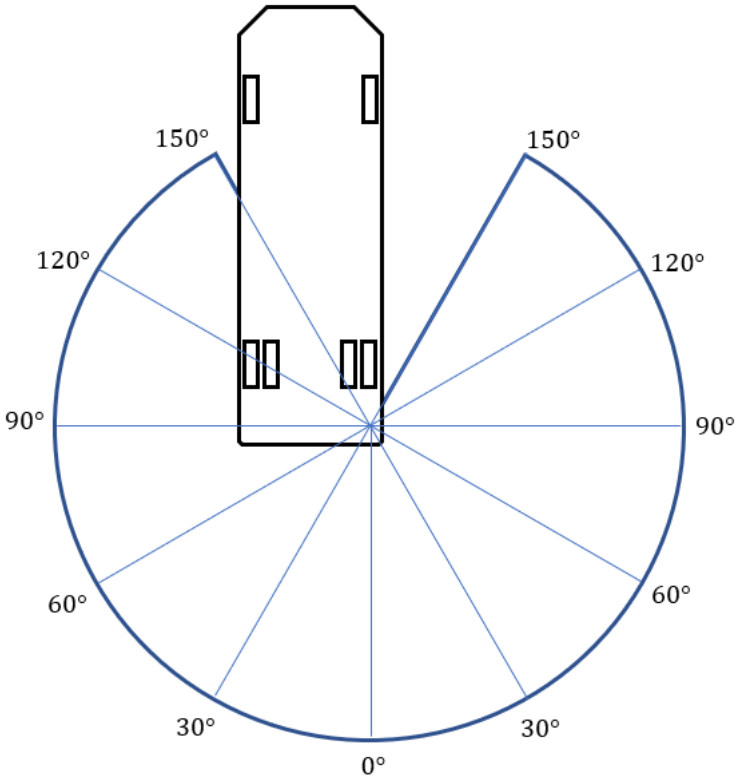

Figure 1: Stability Diagram

To address this problem, we designed a sensor that Stellar can implement to the outside of their trucks that will detect the rotational angle of the arm. This sensor will be able to provide the crane operator with a more precise angle, according to the point of origin defined by Stellar Industries. It will be able to communicate this angle to a handheld controller system used by the crane operators and will display the rotational angle of the truck. The main focus of this project and design is the rotational sensor;

however, Stellar may implement an angular sensor using the same design and making slight alterations to the software.

1.3 System Block Diagram

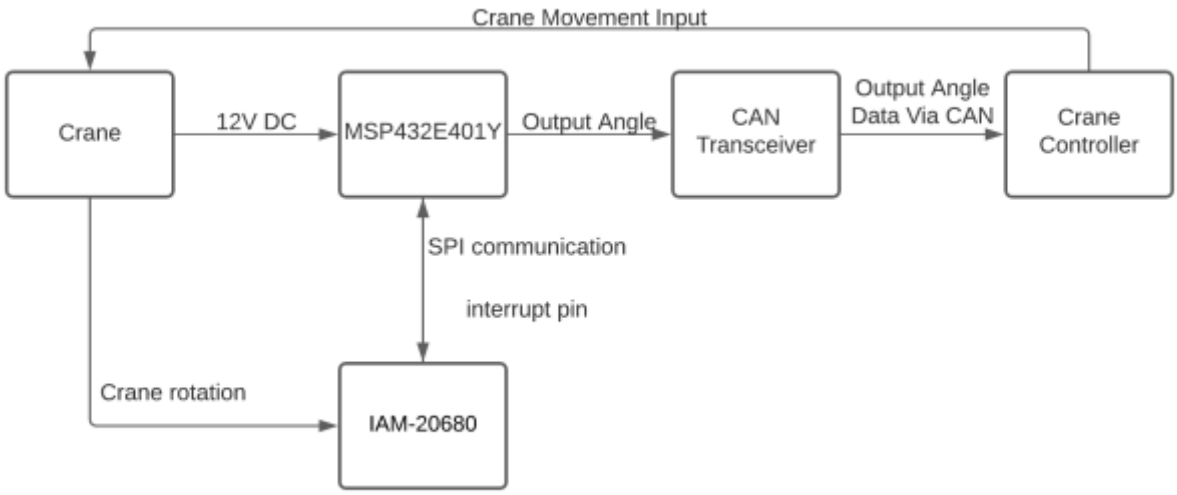

Figure 2: System Block Diagram

1.4 Intended Users and Intended Uses

The intended users for this particular product are the employees of Stellar Industries that are in charge of operating their cranes. They will be able to see an accurate reading of the angle, which increases efficiency and safety. Furthermore, the data gathered from the sensor can be used by Stellar's mechanical department to ensure the cranes are responding accurately to input commands.

1.5 Assumptions and Limitations

Assumptions:

- The end product is for Stellar Industries
- The crane arm will be housed at 180°
- The crane will be operated on flat ground, based on Stellar's instructions for crane operation

Limitations:

- The crane needs to be operated on a flat surface
- The delivered project must be housed to protect the circuit against weather elements

2. Requirements

2.1 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

2.2 Non-Functional Requirements

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

2.3 Technologies Used

The main technologies used in our sensor are SPI for communication from the MCU to the microchip, CAN bus for communication to the controller, and gyroscopes and accelerometers for the calculations of the angle. In this section we will be discussing these technologies and our reasoning behind using them.

2.3.1 SPI

SPI is used to communicate between sensors and MCU's in a wide variety of applications. It is used for short-distance communications and allows for simultaneously sending and receiving data. Due to our lack of knowledge of using SPI or I2C prior to this project, our advisor recommended we use SPI, since he has worked with it and could help us when needed.

2.3.2 CAN bus

CAN bus is a communication system that allows for sending and receiving data over a transmission wire and a receive wire. It is commonly used in vehicles as the main method of communication between various subsystems due to the use of only two wires connecting the entire system. We chose to use this technology as it is the main method of communication between the other sensors currently in use on the crane and the controller.

2.3.3 Gyroscopes

A gyroscope sensor is a type of sensor that measures the angular velocity of an object. It is used in many applications, most notably in phones and aircraft. We chose to use gyroscope sensors for our solution because they are relatively inexpensive and can be easily applied to the outside of the crane, while still remaining accurate.

2.3.4 Accelerometers

An accelerometer is a sensor that measures the acceleration of an object in a particular direction. Accelerometers are often paired with gyroscope sensors, to account for long-term drift that could occur in the gyroscope. We chose to use accelerometers for the reason that they are often packaged with gyroscopes, but also because they would allow us to detect when the crane arm is stationary.

3. Revised Design

Compared to our design from fall semester, our design has been highly altered to fit our needs. Most notably, the parts were exchanged and code was revised.

3.1 Part Changes

3.1.1 Gyroscope and Accelerometer Change

The first version of our design consisted of the MPU-6050 6 axis (Gyro + Accelerometer) IC and the Arduino Uno MCU. This design was developed under the assumption that a gyroscope would suffice for the calculations. During research and testing of this design we found that the gyroscope alone would begin to drift over time. That would cause errors in our calculations. To avoid the error caused by the drifting, we decided to use the accelerometer in unison since that gives better measurements over time. Using the gyroscope, with accuracy in a short amount of time, and accelerometer, with accuracy over longer amounts of time, we were able to produce code that gave us an angle within ± 2° of what we were expecting. The MPU-6050 in this testing setup was placed on the vertical axis and read from the X-axis data of the gyroscope.

For a replacement, our client gave us three choices, with a recommendation for the IAM-20680HP. We talked it over with our advisor and we all came to the same decision to move forward with the IAM-20680HP. This selection allowed us to use SPI communication and has a relatively low cost. The IAM-20680 was also chosen because it has an operation temperature range of -40°C to 85°C, fulfilling our temperature requirements. The IAM-20680HP will also need to interface with a different MCU then we were using. Further details will be discussed in section 3.1.2.

3.1.2 MCU Change

Because we changed our gyroscope, we would be unable to use the Arduino as our MCU. The reasons for changing from the Arduino is expense and the lack of peripherals, mainly SPI and CAN bus. Our client and advisor each gave us a recommendation for a new MCU. Due to the fact that each of us had some experience with the coding style, C, and the program that ran the code, Code Composer Studio, we went with the recommendation of our advisor, the MSP432E401Y TI Launchpad. Our client agreed that would be fine to move forward with. We chose this MCU because it has both CAN bus and SPI support built into the IC. There also was a launchpad version of the MSP432E401Y that allowed us to be able to use the MCU with ease for testing purposes.

3.2 Code Changes

Since last semester, we have only changed the calculation code, as the SPI and CAN bus codes had not been developed at that time.

3.2.1 Calculation

In the initial design of last semester, we used the Dead Reckoning method to calculate the angle, along with a high-pass filter and an initial drift correction. Instead of determining scaling factors for the time interval and the bits-to-degrees conversion, we used trial-and-error to scale the outputs to (rough) degrees. We also incorrectly concluded that this method depended on the mounting radius of the sensor.

Because of this, at the beginning of this semester we began to formulate an alternate method. This secondary method would rely on more sensory information: the X, Y, and Z accelerometer data, as well as the X and Y gyroscope data. All this would be combined using the formula below.

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

where α and □ are the rotations about the X- and Y-axis, respectively. This formula showed very little drift in MATLAB testing and compensated for any tilt of the truck that might be present. While this method was incredibly accurate, it would mean that the sensor would have to be mounted in such a way that kept its rotation rigid to the body of the crane, not to the arm. It was then found that the Dead Reckoning method indeed did not rely on the radius of mounting being known. Because of this, the team decided to return to the former method of calculating the rotational angle of the crane.

Using the gyroscope sensor, we can determine the angular velocity of the truck. By integrating these measurements over time, we can determine relative movement of the truck. And since the crane arm will always start at 180°, the stored position of the arm, we can determine the absolute position of the truck. We picked this method because it only required measurements on a single gyroscope axis, which can increase the sampling rate, and because this method would not rely on knowing the mounted radius of the sensor unit.

3.3 Relevant Standards

The standards that applied to this project were mostly those of communication between units. More specifically, both SPI and CAN bus standards were researched and implemented in this project.

4. Implementation

Because of the hardware changes at the beginning of the second semester, most of our implementation occurred during the second semester. To accomplish our goal, we divided our work into separate tasks: SPI programing, CAN bus programing, and calculation programing.

4.1 Sensor Hardware Overview

Figure 3: Pinout Block Diagram

Figure 3 shows the pin connections of the pins on the MSP432E401Y to the IAM-20680HP. We chose to use the pins we are using because PQ is the SPI inputs and outputs and PA0/PA1 are the CAN bus receive and transmit respectively. PK6, PK7 and PP4 are GPIO pins on the MSP432E401Y that connect to other pins on the IAM that are needed for functionality. The interrupt pin on the IAM will be used to trigger a flag to allow for data to be sent over CAN bus, this topic will be further discussed in section 4.3 CAN bus.

4.2 SPI

The SPI is used to transmit addresses and data along with receiving data from a specific address. For the SPI to be used it needs to assign a variable to the Handle, Parameters, and a Transaction function. These functions are provided by the TI header files in Code Composer Studio. The Handle is used to determine if the SPI is initialized or not, but for the SPI to be initialized the Parameters have to be set first.

When setting the Parameters, there are 7 that need to be addressed. Those consist of the transfer mode, transfer timeout, transfer callback function, mode, bit rate, data size, and frame format.

- Transfer mode
    - Tells if the SPI is in blocking or callback mode.
- Transfer timeout
    - Tells SPI to wait for how many system ticks
- Transfer callback function
    - Used only when in callback mode
- Mode
    - Select master or slave
- Bit rate
    - Selects SPI bit rate in Hz (Max = 60 MHz)
- Data size
    - Selects data frame size in bits
- Frame format
    - Selects the polarity and phase of the clock to transmit the data

For our application, we are using blocking mode, since we want to stop code execution until the transaction is complete. We are using a macro given to us in the header files called SPI_WAIT_FOREVER for the transfer timeout parameter. The transfer callback function we are assigning NULL since we do not want to use the callback mode. When selecting the mode, we chose the MSP432E401Y to be a master, since we need to read and write values to the TDK-20680HP. The bit rate we chose to be 100KHz. For data

size we chose to send 8 bits to make sure we kept track of each address and data point.

Once the Parameters were set, we could open the SPI, allowing for us to use the Transaction function. To use the Transaction function we had to set the count, txBuf, and rxBuf.

- Count
    - Sets the number of frames for a transaction
- txBuf
    - Is a pointer to a buffer with data that is to be transmitted
- rxBuf
    - Is a pointer to a buffer with data that is to be received

We chose count to be 1, which allows us to send 1 transaction of 8 bits since the data size is set to 8. The txBuf and rxBuf are set to indices in an array for either transmitting or receiving values. When transferring data through the SPI we have to make sure that we are sending the correct address and data to the TDK-20680HP for proper set up and usage.

4.3 TDK-20680HP

4.3.1 SPI

The protocol involving communication via SPI is done via two bytes. The first bit of the first byte communicates whether we want to read (1) or write (0). The next seven bits of the first byte contain the address of the register. For instance, if we wanted to read from the register at address 0x3B we would send 0b1011_1011 or 0xBB as our first byte. The second byte would contain the data to be sent.

4.3.2 Registers Needed

The TDK-20680HP has specific registers that are required to be set up before the sensor can produce values. Provided in the data sheet, there is a single register that needs to be configured before any others since it will provide power to the rest of the sensor. The first register that needs to be configured is called ‘Power Management 1’. This register will be sent 0x81, which allows it to reset the internal registers and restore them to their default values and also selects the best available clock source. After initializing Power Management 1, we can use the rest of the registers below.

Registers from TDK	HEX ADDRESS	READ/WRITE
108 PWR_MGMT_2	6C	Write
27 GYRO_CONFIG	1B	Write 250 dps
28 ACCEL_CONFIG	1C	Write 2g
29 ACCEL_CONFIG_2	1D	Write (8 Samples)
59 ACCEL_XOUT_H[15:8]	3B	Read
60 ACCEL_XOUT_L[7:0]	3C	Read
61 ACCEL_YOUT_H[15:8]	3D	Read
62 ACCEL_YOUT_L[7:0]	3E	Read
63 ACCEL_ZOUT_H[15:8]	3F	Read
64 ACCEL_ZOUT_L[7:0]	40	Read
71 GYRO_ZOUT[15:8]	47	Read
72 GYRO_ZOUT[7:0]	48	Read

Table 1

4.4 Angle Calculation

In the angle calculation section of our code, first we calculate the offset of the gyroscope. Since gyroscopes sensors are prone to drift over time, 200 samples of gyroscope output are taken and averaged together while the system is not in motion. This value is then divided by the sensitivity scale rate of the gyroscope, as defined by the datasheet. For our sensitivity scale rate of ±250 dps, this value is 131 LSB/dps.

After the drift has been calculated, and once the crane starts moving, our program will collect data 10 times per second. The data received will be the angular velocity of the crane. After converting this to degrees per second (by dividing by 131 LSB/dps), we then subtract our offset value from our received data. This should account for most of the drift, but not necessarily all.

To compensate further for the drift, a high-pass filter will be implemented. A high-pass filter will allow higher-read values to pass freely, but sets the read velocity to 0 if it is below a certain threshold. This will mitigate additional drift that might be present. The corner value of the filter can only be determined with testing on the actual crane.

After this, the velocity is multiplied by the amount of time elapsed. This is nominally 0.1 seconds, but to be more precise, the clock difference between starting the calculation and this step is calculated, and is the scaling factor for the velocity. This scaling will also produce a positional change. Finally, this positional change is added to the current angle of the truck, which initially starts at 180°.

4.5 CAN bus

For CAN bus, we are sending the output data from our angle calculations discussed in section 4.3. To set up CAN bus we are using a modified version of the example code from the MSP432E401Y DriverLib. We added functionality to set a flag when the interrupt pin connected to PP4 is triggered. This flag will then allow for the calculated angle to be sent via a CAN bus message with a size of 16 bits.

As for setting up the CAN bus functionality, we set macros for both the system clock and CAN address. We set a macro for the system clock to avoid having to set the system clock multiple times throughout the initialization process. We used a macro for CAN address because it can vary from the address we have been using for testing from the address in Steller's crane mounted trucks.

To configure CAN we used the example from the driver lib to guide us. We changed the pins used for CAN to be PA0 and PA1 as we are using UART2 for CAN communication. As for our UART configuration, we again based our code off of the examples found in the driver lib, changing the baud rate to be 115200 to match with our computers for testing purposes.

For our main program we set the message data to be the calculated angle value and set pin PP4 as an interrupt. From there we ran the configure UART and CAN functions along with setting the CAN message address. When the interrupt is triggered, the interrupt handler will set a variable named txMsg to true allowing the message to send using an if statement. In the if statement we used the MAP_CANMessageSet function set and send our message. We also included UARTprintf to display the message and if it was sent or not for testing purposes.

4.6 Addressing Common Concerns

Throughout CPRE 492, a couple of concerns were raised regarding our approach to solving this problem. As they might be concerned that others hold, and did not particularly fit in any other section, we thought it best to address them here.

Will the sensor output correctly on uneven ground?

Stellar Industries explicitly tells its crane operators to find the flattest ground possible before attempting to operate the crane. In training examples created by Stellar, the crane operators were warned about placing the truck on slopes or uneven ground. Since the gyroscope is detecting the rotation of the board about only one axis, the sensor does not need to be perfectly flat to operate accurately.

Will the sensor output correctly, given the vibration of the crane?

The gyroscope sensor will only be powered and measuring while the truck is stationary. To compensate for the vibration of the truck while it is stable, a high-pass filter should be applied that would mitigate the larger frequencies of the truck's vibrations.

Will the gyroscopic drift be too much for a ±5° accuracy?

The ±5° accuracy was not given to the team by Stellar Industries, but instead was decided by our team as a goal. That being said, the system will recalibrate itself every time the crane restarts by determining what the gyroscope reads while the truck is not moving. This static gyroscope drift is taken into effect with each measurement from the sensor.

5. Testing

5.1 SPI Testing

Using an oscilloscope, we obtained traces to make sure that our SPI functionality on the MSP432 was sending accurate data to the TDK sensor. We sent 0x81 (0b1000_0001) through the SPI and put it in a while(1) loop to make sure that it was sending recurring and consistent readings of 0x81. The yellow trace represents the data that will be sent to the TDK and the green trace represents SPI clock.

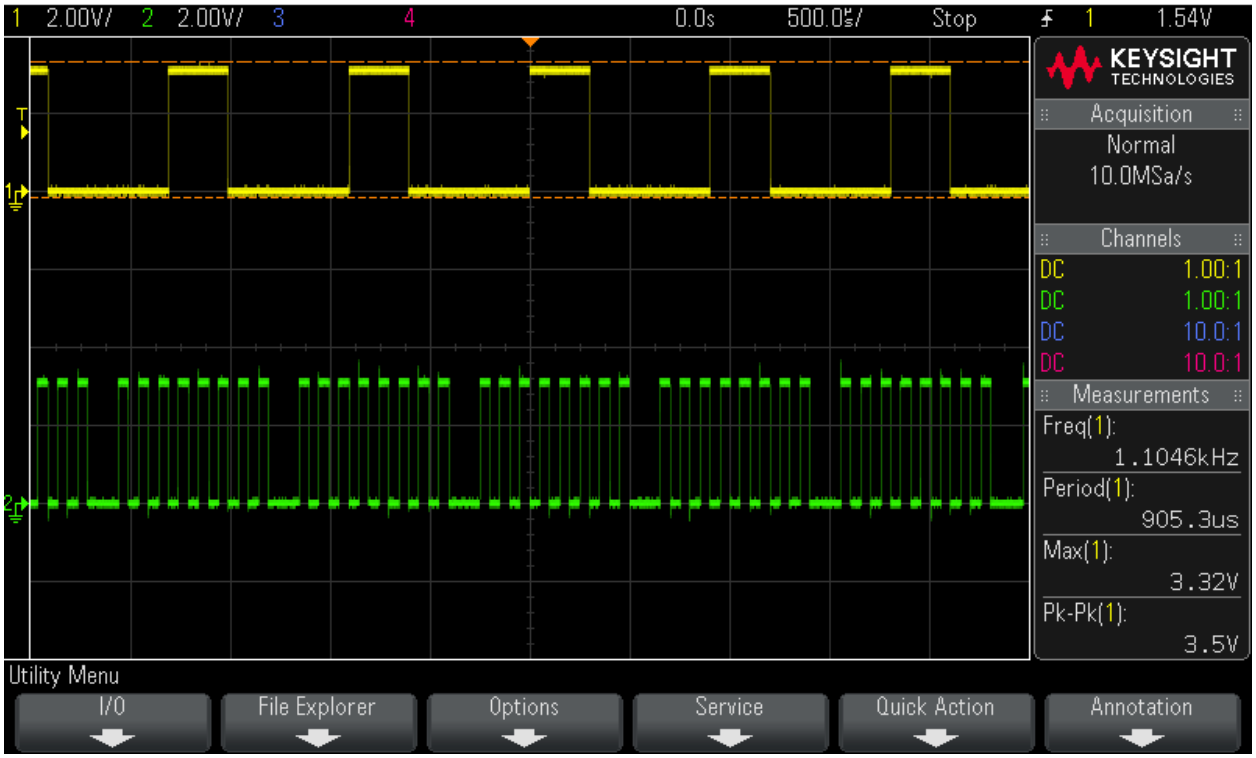

Figure 4: 0x81 sent with a polarity of 0 and phase of 0

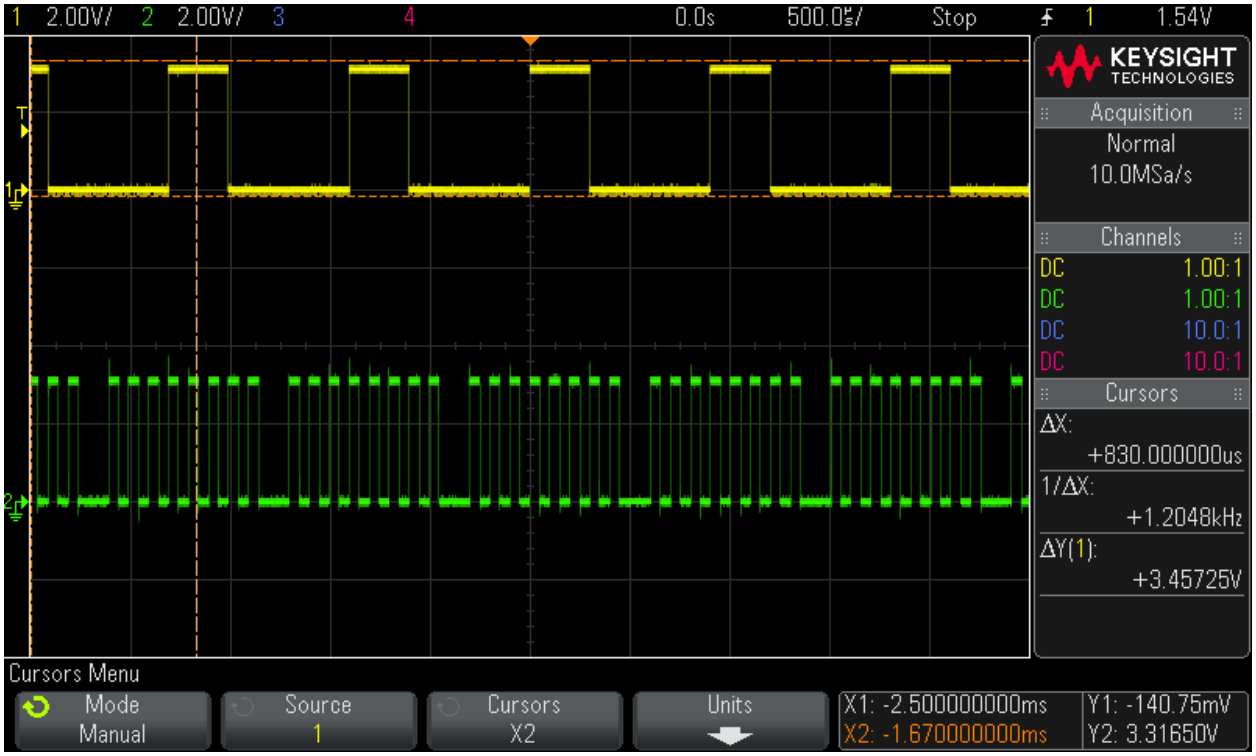

Figure 5: 0x81 sent with a polarity of 0 and phase of 1

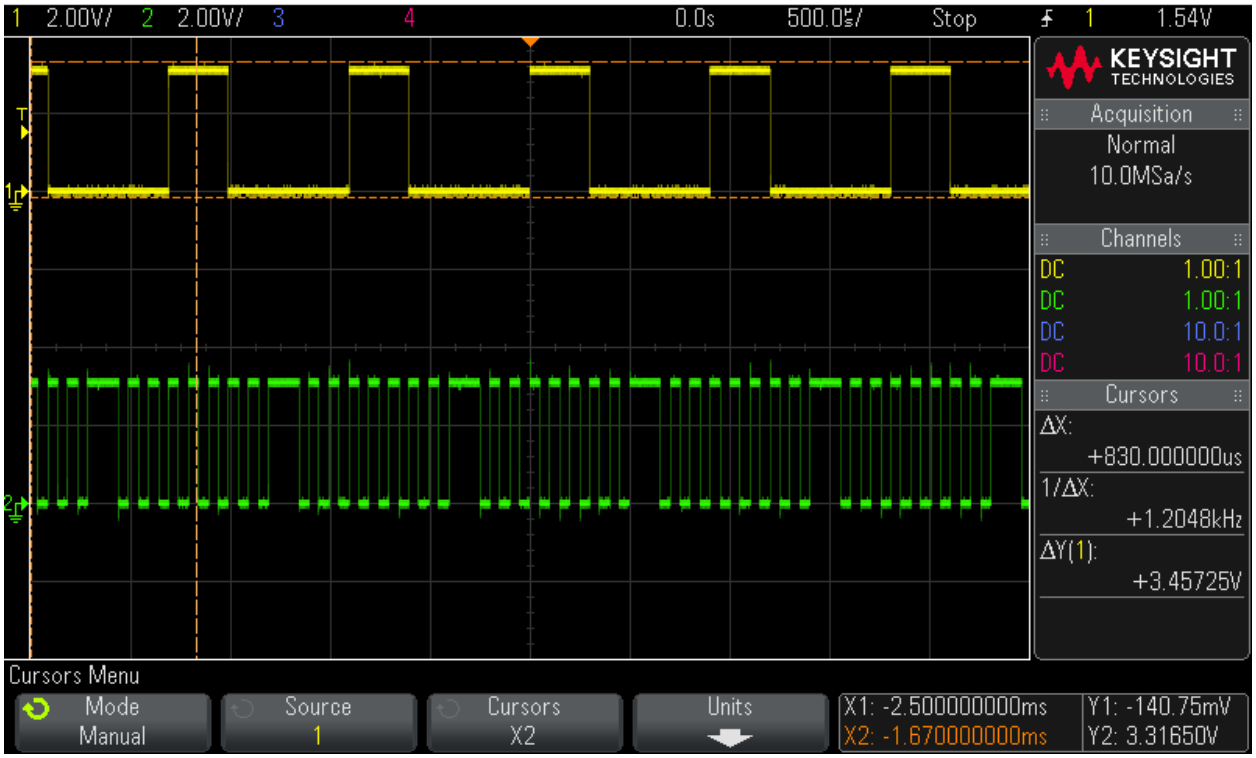

Figure 6: 0x81 sent with a polarity of 1 and phase of 0

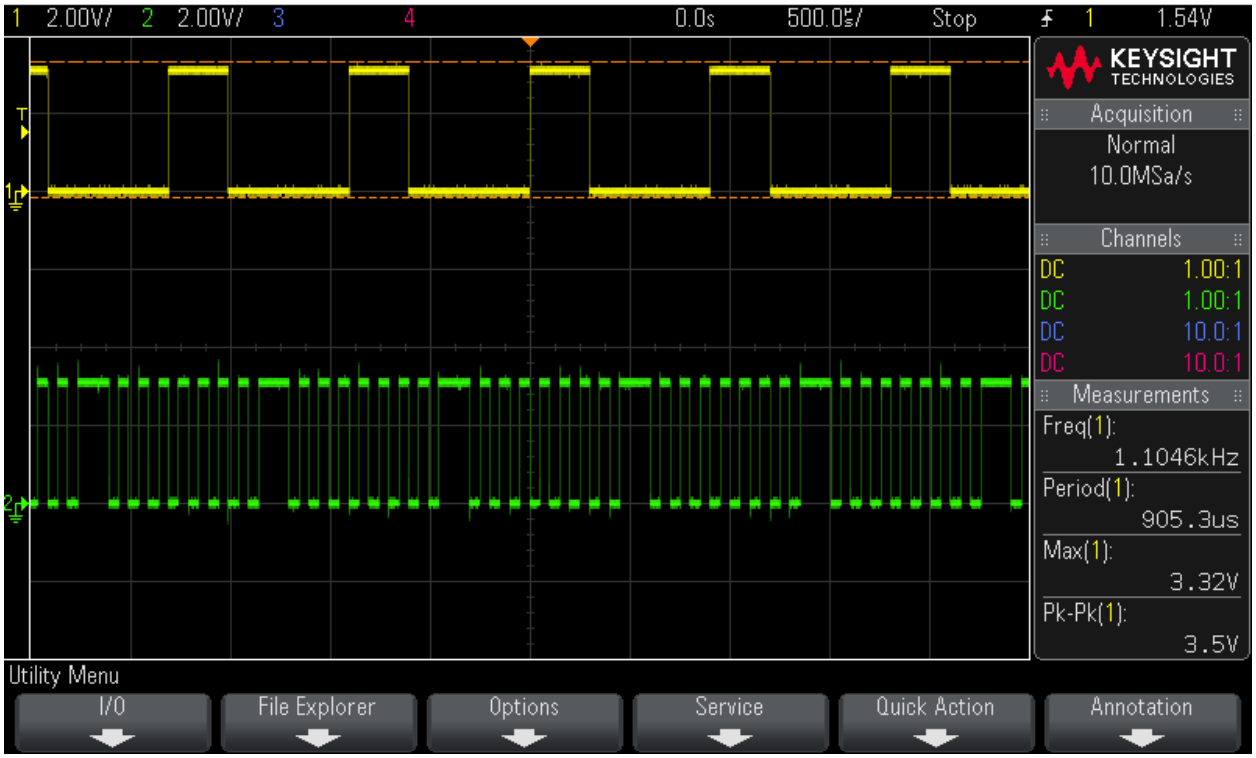

Figure 7: 0x81 sent with a polarity of 1 and phase of 1

It can be observed that the data being sent is the same but, by manipulating the phase and polarity, we are able to see their impact on the SPI Clock. The polarity bit makes sure that the clock starts with a 1 bit every time and the phase of 1 shifts the clock slightly forward.

5.2 Calculation Testing

The testing for our calculation process was done mostly on our preliminary board, the MPU-6050 as well as an Arduino Uno. The testing predominantly carried out was for determining the accuracy of the system over time.

For testing the accuracy of the system over time, the sensor was left calculating the angle for an hour. Every five minutes, the angle of the system was changed. At the end of the testing period, the gyroscope acquired a 12° drift. After implementing a high-pass filter and repeating the test, this drift was mitigated to 3° after an hour of use.

5.3 CAN bus Testing

To test CAN bus we used two MCUs, one flashed with our CAN bus code and the other with the can_single_message_receive_MSP_EXP432E401Y_nortos_ccs program found in the driver lib of our MCU. For testing if our messages were being transmitted, we needed to use a CAN transceiver test board to allow us to convert our messages

into CAN high and low, along with reading the CAN high and low messages. We chose to use the TCAN1042DEVM dev board from Texas Instruments as our CAN transceiver because of the wide array of options available when testing.

For testing we set an LED on the MCU dev board that will light up when the PP4 interrupt is triggered. This allows us to know if the interrupt flag has been triggered debugging if it has not. We are also using UART to print the message we are sending and give us feedback on if the message was sent or not. Again, this is useful for us as we will then know if an error occurred in either our set up or in our code. On the receiving MCU we are using UART to view any CAN bus messages being received. Again this is useful for debugging as we can check if the message being received is in the proper format and that the message is intact.

Overall our results have come back positive, at first we had issues with the baud rates not being properly configured but that issue was quickly fixed. We were able to send and receive data from our CAN bus program to the driver lib CAN bus receive program.

6. Closing Remarks

At this point, we feel we have successfully designed a base for the rotational sensor and feel ready to hand the project over to Stellar Industries for any additional modifications. While we did not finish the project ourselves, we still believe our work has been beneficial to our clients, and that our solution is the best one for Stellar Industries.

Appendix I: Operation Manual

This section will cover how Stellar Industries will need to install and set up the sensor unit in order to correctly calculate the angle of the truck.

1. Modifications to Code

To ensure that the unit is working in the intended manner, modifications will need to be made to the provided code.

1. If the sensor is not planned to be mounted with the largest face pointing up or down, the specific register accessed by the MSP432E401Y from the IAM-20680 will need to be updated. Initially, the sensor is set to measure rotation along the Z-axis, or if the sensor is mounted such that its largest face is pointed up or down.
2. If the user wishes to increase the accuracy of the system, a corner value may be inserted into the high-pass filter of the system. This value will need to be inputted as the peak measured value of the system while the crane arm is at rest and after the calibration setup has already run.
3. If the user wishes, an additional implementation of an angular sensor may be implemented without any additional software. To access this angle measurement, the user will simply need to read from the gyroscope that rotates in the correct axis of the angular movement, and repeat the same calculations laid out in the code on the new gyroscope data.
4. The user will also need to check the CAN address of the system and make sure it is the same in the code. If the two addresses are not the same, the user will need to change the macro named CAN_ADDRESS to the correct address value and reflash the program on the MSP432E401Y.

2. Installation

To ensure the unit is properly installed on the crane, see that the following conditions are met:

1. The MSP432E401Y will need a step-down transformer from 12V to 3.3V
2. The IAM-20680 and MSP432E401Y should be placed in a housing unit. The housing unit may be placed anywhere on the arm of the crane without impinging the functionality; however, if the sensor is not facing straight up or down, the gyroscope axis direction must be modified in the code. (See Section 1)
3. To set up CAN bus the user will need to connect the transmit wire to pin PA1 and the receive wire to pin PA0 on the MSP432E401Y.

3. Notes During Use

If the sensor becomes off and needs to be recalibrated, the system should be restarted while the arm is at 180°. If the arm is still and the angle displayed is increasing or decreasing, a high-pass filter should be applied (See Section 1).