**\*\*Please turn in this sheet during your Judge Interview along with your Engineering Notebook\*\***

|  |  |
| --- | --- |
| **Team #3543** | **Team Name: Titan Robotics Club** |

**Autonomous objectives:**

Our objectives are to provide an autonomous that performs the tasks very accurately and reliably utilizing sensors and advanced algorithms. It also provides flexibility and fault tolerance by providing options to our autonomous OpMode. We are using a set of menus that prompt the driver for different choices. For example, the driver can choose what color alliance we are in (2 variations: red, blue), whether we should delay starting our autonomous so we don’t bump into the robot of our alliance partner, which balancing stone our robot is starting on, whether we will attempt to knock off the opponent’s color jewel and whether we will attempt to put the glyph into the appropriate column of the crypt box. Detail description with diagrams of each strategy can be found in a later section. The autonomous strategies are listed below:

1. **Do Nothing** – No autonomous. This is useful when our robot is malfunctioning and cannot perform any autonomous.
2. **Full Auto** – This is the one autonomous that does everything with five different parameters selected by the driver via our menu system:
   1. Alliance – Chooses between RED or BLUE alliance.
   2. Delay – Chooses the amount of time to wait before starting robot movement. This is useful for coordinating with our alliance partner in case their autonomous may cross path with ours. By delaying some amount of time, we can avoid potential interference with our alliance partner’s autonomous.
   3. Start position – Chooses which balancing stone our robot is on so it knows which crypto box it will deliver the glyph to.
   4. Do jewel – Chooses whether the robot will attempt to detect the jewel color and knock off the opponent’s color jewel. This option allows us to disable this task in case our color sensor is not functioning properly.
   5. Do crypto box – Chooses whether the robot will use Vuforia to determine the crypto box column to place the glyph. This option allows us to disable this task in case Vuforia directed navigation is malfunctioning.
3. **Distance Drive** – If all other sensors are failing but the encoder and gyro are still functioning, this option allows our robot to drive off the balancing stone by specifying the driving distance.
4. **Timed Drive** – If all sensors are failing, this option allows our robot to just drive the robot off the balancing stone by time.

**Sensors used:**

* **Motor encoders**: Our robot uses encoders on the mecanum drive base to measure the distance traveled by the robot in both X and Y directions. Our software is capable of combining the four encoder values in a way to accurately report the X and Y distances traveled by the robot.
* **Touch sensor**: Our robot uses many touch sensors as limit switches. We use one pair to set the lower and upper angle limits of the relic arm elbow so that the relic arm will be operating within its mechanical limits. We use another pair to set the extend/retract limits of the relic arm. Finally, we use one limit switch for the lower limit of our glyph elevator. It stops the elevator motor from going below the lowest height of the elevator. Since the elevator can go relatively high, there is no good way to mount an upper limit switch. For this, we use a string potentiometer. See below.
* **String Potentiometer**: We use a potentiometer on a spring loaded retractable string system (<http://www.andymark.com/product-p/am-2673.htm>) that ties to the top of the elevator. When the elevator goes up, it pulls the string and turns the potentiometer. When the elevator goes down, the spring will wind the string back turning the potentiometer. It provides very accurate absolute position information on the elevator height. We also use this height information to limit the height the elevator can travel in essence acting as a software upper limit switch.
* **Gyro sensor**: Keeps track of robot heading for precision turn and running straight. Our turns are very precise because we use absolute heading to eliminate cumulative error. We are using the built-in IMU in the REV expansion hub for keeping tracking of the robot heading.
* **Color sensor**: Tells us the color of the jewel. We read the RGB values from the color sensor and convert them to the HSV color space. This allows our jewel color detection algorithm to be largely independent of lighting conditions so we can still tell the jewel color accurately with different shades of blue and red.
* **Ultrasonic sensors**: We use two Modern Robotics Range sensors, one on the left side and one on the right side. The left and the right ultrasonic sensors allow the robot to accurately strafe to the correct column of the crypto box for the placement of the glyph.

**Key algorithms:**

**PID** (**P**roportional/**I**ntegral/**D**ifferential) control:

* Instead of using the motor controller built-in PID control which is doing individual motor control, we used overall PID control across all wheel motors which maintains power distribution between them; combined with motor encoders, this allows our robot to shift power across motors if one is moving faster than the others, preventing the robot from unintentional turning.
* Our PID controlled drive algorithm consists of three software PID controllers: one to calculate net power output for the X direction, another for the Y direction and one for maintaining the robot heading. By combining these three PID output, we are able to distribute power appropriately to each wheel and accurately navigate the robot to the desired target position. This cannot be achieved by using motor controller built-in PID control because individual motor PID control does not take into account of other motors.
* Our PID controlled drive can be used not only with encoders and gyro, it can be used with any sensors for different purposes. For example, if the X PID controller uses an ultrasonic range sensor to control the distance from the wall, the Y PID controller uses the encoders to move the robot parallel to the wall in the Y direction and the turn PID controller uses the gyro to maintain the heading, then we have a wall follower. Another example: if the Y PID controller uses the ultrasonic range sensor to stop the robot when it’s too close to the wall and the turn PID controller uses the light sensor following the edge of the line by adjusting the turn of the robot, then we have a line follower.

**Gravity compensation**:

Our earlier version of the relic arm has a NeveRest motor controlling an elbow. This kind of design is tricky because the load on the elbow is not linear. The load reaches maximum when the arm is horizontal and zero when the arm is vertical. In between, the load varies proportional to the cosine of the elbow angle. We attempted to do PID control on the elbow but PID control is only good if the load is relatively linear. Therefore, we have developed an algorithm to compensate for gravity effect on the arm making it appears to be linear to the PID controller. Here is the algorithm we used in our code to compensate for gravity. Power calculated from Gravity Compensation is added to the PID motor power output.

*//  
// To counteract gravity, we need to add power compensation to the elbow*

*// motor. We are using a NeveRest 60 motor. The performance spec of this*

*// motor is:  
// - Stall Torque: 593 oz-in  
// - Output counts per revolution of Output Shaft (cpr): 1680 Pulses*

*//  
// The greatest effort to hold the elbow is when the arm is horizontal.  
// Arm length = 19 inches.  
// Weight of arm at 19-inch from fulcrum = 21.16 oz.  
// Torque at fulcrum = Weight of arm \* arm length*

*// = 21.16\*19 = 402.04 oz-in.  
// Additional gear ratio = 2:1.  
// Torque at motor = 402.04/2 = 201.02 oz-in.  
// To maintain arm at horizontal position, we need to apply*

*// 201.02/593 = 0.339 of full power.*

*//  
// To calculate Degrees/EncoderCount, one revolution of the motor will*

*// give 1680 counts.  
// One revolution of the motor will yield 180-degree movement of the arm*

*// because of the 2:1 gear ratio.  
// Therefore, the degrees\_per\_count = 180.0/1680.0.  
// The arm is resting at 40 degrees below the horizontal position. A lower*

*// limit switch will clear the encoder when the arm is at rest. So the*

*// scaled position read from the encoder should subtract 40 degrees from*

*// it.  
//*

**PID Controlled Elevator:**

Even in TeleOp, our elevator is controlled by PID. Although we have a lower limit switch that will cut motor power when the elevator reaches the bottom, running full power going down may destroy the elevator even if it hits the lower limit switch. In order to prevent this kind of damage, we use PID assist Teleop control. In this scheme, we set the PID setpoint to either the minimum or the maximum elevator height depending on whether the elevator is going down or up. By doing so, even if the operator is pushing the game controller stick full power going down or up, the elevator will slow down when it is approaching the setpoint. If the operator is moving the elevator slowly by gently pushing the game controller stick, our algorithm will use the game controller stick value as the clamping value on the PID output power. For example, if the elevator is going up and is far away from the maximum height, PID may calculate the motor output power to be full power but since the game controller stick is pushed only half way, we will clamp the motor output power to only 50%, thus achieving the effect of user controlling the elevator speed and yet PID will protect the elevator by slowing down when closing to the limits.

**Driver controlled enhancements:**

One of the unique features that we have developed is a menu driven autonomous selection. As we have mentioned, we have 4 autonomous strategies and one of which has 5 options. This creates about 19 different variations. Without this menu selection system, we would have to create 19 OpMode programs. Not only making it tedious to write, it also makes it very difficult to maintain. Most of the OpModes would have very similar code with only minor differences. Tuning all OpModes will be a nightmare. Instead, we have only three OpMode programs: Autonomous, TeleOp and Test. When running the Autonomous OpMode, the driver is presented with various menus displayed on the Driver Station forming a decision tree. The driver will make choices using the gamepad controls to scroll through options. Once all the choices are made, the selected strategy will be run with other selected choices as parameters. The diagram in the next section shows the organization of this menu tree.

Another unique feature is our Test OpMode. It serves two purposes: diagnostic and calibration. If the robot is not functioning properly, it allows the driver to diagnose the root cause of the problem. When run, the Test OpMode also presents a menu of diagnostic tests to the driver:

* **Sensors Test**: This test reads every sensor on the robot and display their values on the Driver Station in real time. While this test is running, the driver has full TeleOp control so he/she can drive the robot around as well as test every subsystem.
* **Drive Motors Test**: This test runs each of the drive motors on the drive base sequentially for 5 seconds each with the same power and displays the encoder count of each motor at the end so that if the encoder values have major differences, it could mean some of the wheels have mechanical problem such as friction, chain tension or weak motor etc. We can also verify if the motors have correct directions with this test.
* **X Timed Drive Test**: This test runs the drive base in the X direction for the selected number of seconds. It is primarily used for calibrating the INCHES\_PER\_ENCODER\_COUNT scaling factor for the X direction. At the end of the run, encoder values are displayed on the Driver Station and one can measure the distance traveled using a tape measure. The scaling factor is determined by dividing the measured distance with the encoder count.
* **Y Timed Drive Test**: This test is similar to the **X Timed Drive Test** but for the Y direction.
* **X Distance Drive Test**: This test runs the drive base in the X direction for the selected distance. It is primarily used for calibrating the PID constants for the X PID controller. One can adjust the PID constants to make the robot go as close to the target distance as possible without oscillation and within tolerance.
* **Y Distance Drive Test**: This test is similar to the **X Distance Drive Test** but for the Y direction.
* **Gyro Turn Test**: This test rotates the drive base for the selected number of degrees. It is primarily used for calibrating the PID constants for the Turn PID controller. One can adjust the PID constants to make the robot rotate as close to the target heading as possible without oscillation and within tolerance.
* **Left Range Drive Test**: This test runs the robot in the X direction using the left ultrasonic range sensor towards or away from the wall for the selected stopping distance. It is primarily used for calibrating the PID constants for the X Range PID controller. One can adjust the PID constants to make the robot go as close to the target wall distance as possible without oscillation and within tolerance.
* **Right Range Drive Test**: This test is similar to **Left Range Drive Test** but using the right ultrasonic range sensor instead.

All these features are supported by our library and is available on GitHub as part of our Open Source Software. (<https://github.com/trc492/Ftc2018RelicRecovery>)



**Engineering notebook references:**

|  |  |
| --- | --- |
| **Feature** | **Notebook Pages** |
| Autonomous Strategies | 143-150 |
| Sensors | 129-134 |
| PID Control |  |
| Choice Menu System | 142 |
| Test OpMode | 140 |

**Autonomous program diagrams:**

The following autonomous diagram shows our robot on all four possible starting positions.



**Blue Near:**

1. Read the pictograph using Vuforia. Detect the jewel color using the REV color sensor. Knock off the jewel with the opponent’s color. Grab and raise the glyph and go forward to move off the balancing stone.
2. Strafe to the right to align the crypto box column according to the pictograph.
3. Lower and release the glyph on the floor and push the glyph forward into the crypto box column.
4. Back up a little so that the robot is not touching the glyph but still within the safe zone.

**Blue Far:**

1. Read the pictograph using Vuforia. Detect the jewel color using the REV color sensor. Knock off the jewel with the opponent’s color. Grab and raise the glyph and go forward to move off the balancing stone.
2. Turn left 90 degrees.
3. Strafe to the right to align the crypto box column according to the pictograph.
4. Lower and release the glyph on the floor and push the glyph forward into the crypto box column.
5. Back up a little so that the robot is not touching the glyph but still within the safe zone.

**Red Near:**

1. Read the pictograph using Vuforia. Detect the jewel color using the REV color sensor. Knock off the jewel with the opponent’s color. Grab and raise the glyph and go backward to move off the balancing stone.
2. Turn right 180 degrees so the front of the robot is facing the crypto box.
3. Strafe to the left to align the crypto box column according to the pictograph.
4. Lower and release the glyph on the floor and push the glyph forward into the crypto box column.
5. Back up a little so that the robot is not touching the glyph but still within the safe zone.

**Red Far:**

1. Read the pictograph using Vuforia. Detect the jewel color using the REV color sensor. Knock off the jewel with the opponent’s color. Grab and raise the glyph and go backward to move off the balancing stone.
2. Turn left 90 degrees so the front of the robot is facing the crypto box.
3. Strafe to the left to align the crypto box column according to the pictograph.
4. Lower and release the glyph on the floor and push the glyph forward into the crypto box column.
5. Back up a little so that the robot is not touching the glyph but still within the safe zone.