Engineering Journal

FTC Team 4278
de.evolution

November 15, 2013

Abstract

FTC Team 4278 proposes the creation of a robot to address all aspects of the FTC 2013-14 competition, "Block Party." The proposed design has the capacity to address, in the teleoperated period: (1) retrieval of blocks; (2) placement of blocks into the block crates; (3) raising the FTC flag; (4) hanging against the field bar. In addition, during the autonomous period, the robot will address the following elements: (1) placement of blocks in the correct bin, as indicated by an IR beacon; (2) maneuvering to the ramp.

In order to achieve these goals, it was determined that the robot should: (1) be very difficult to push; (2) be fast and reliable in movement; (3) have a low center of gravity; (4) have as few unique components as possible; (5) be as compact as possible; (6) pick up blocks with at most one motion; (7) place blocks in crates with at most one motion; (8) hang with at most either two motions on an existing component, or one motion on a new component; (9) be able to raise the flag with one component; (10) be electrically stable, generating as little static as possible.

Goals set for autonomous include: (1) being able to score in the crate denoted by the IR beacon; (2) being able to travel to the nearest ramp; (3) being able to address and recover from any problems which may arise on the field, including, but not limited to, obstructions caused by other robots; (4) potentially being able to score both autonomous blocks. For (3), the intent is to travel onto the ramp regardless of potential obstruction by other robots. Decision logic is used to avoid obstructions.

Over the course of this engineering document, we address these problems, determine how we solve them, and explain the process which lead to these conclusions. Most notably, the development of a prototype robot allowed us to identify critical problems with our design and address them before implementation.

Additionally, the code which goes into this design will be explained.

Contents

1 Preseason Overview - Meeting 1: 2013-08-31

We held a preseason meeting in order to go over scheduling, recruitment of new members, and hold a quick review of what we learned from our last season. One of the main items we discussed was our scheduling, and as a team we decided to take an aggressive approach towards our first couple of regional competitions in order to secure a place in St. Louis. Because of this, we may need to cut back on scoring the maximum number of points and instead focus on scoring a high, yet consistent number of points. We found out that we need to secure our electronics and work with the field control issues that we were issues. We also plan on placing a much larger emphasis on the CAD design of our robot than we have in our two previous years as it is an efficient way to quickly discover and troubleshoot problems prior to building the system.

Our final goal is to have two weeks of drive practice before our first regional on December 14th, important for both driver and coaches, to figure out the timing of the game, as well as things that we can or cannot do in game. Organization of the team this year will be facilitated through the use of Google Groups, which will allow all the team members and their parents to be easily contacted for meetings, and will hopefully foster some discussion over build design or game strategy.

We also decided to meet a few hours after the game was released next week so team members could think about strategies ahead of time and add to the strategy discussion of our first meeting of the season. The meeting was fairly short, but got the team in the right mindset for the upcoming season, and got everyone excited for the new season!

2 Initial Design – Meeting 2: 2013-09-07

The team decided last week to meet a few hours after the game video and rules were released, so by the time our meeting started, everyone had an idea of how they thought the robot should work, and be built. Fortunately, most team members were on the same page and wanted to focus on scoring as many blocks into the baskets as possible, in a manner that would allow for a integration in the lifting and scoring mechanism. We decided to focus on every aspect of the game for our qualifiers, since it seems like we could consolidate all of the mechanisms into very few.

After a fairly long strategy discussion, some ideas were thrown around as to how to pick up and score the blocks, since we always have a difficult time getting our game pieces, and a rough idea of a mechanism was developed that would use a roller system and a block hopper that would pick up and score the blocks. There is still significant discussion considering the way we will go about constructing a lifting mechanism, with different arguments for and against a rotating arm and a more standard but perhaps less efficient forklift, similar to "Ring It Up!". Some of our team members have some experience with constructing forklifts from prior FTC seasons, so we have some idea of what kind of issues we might come across with that kind of design. One thing we discussed was making sure we either construct our purchase our materials with a little more regard for quality than we have in previous years, but we wanted to focus on simplicity this season as it worked out very well for us three years ago, and the ideas we are thinking of currently all seem to fit this idea.

Members of the team have assumed different jobs to complete before the next meeting, such as researching lifting mechanisms, getting a BOM for the field and purchasing the materials, and researching the specifications of an IR beacon and sensor, since the team also decided that scoring the block during autonomous is absolutely imperative toIf a material is homogeneous then the tensile and flexural strengths are identical. However, most materials have defects in them which act to concentrate the stresses locally, which in turn cause a localized weakness. Our group opted to use round hardwood toothpicks that are cut with the grain as they provide the strongest and most rigid support system. Other toothpick styles, such as flat types, were considered but we opted against them due to their inability to survive in the presence of minimal force. I have determined that the flexural strength is greater in the rounded hardwood toothpicks over the flat toothpicks as follows. We first see that method for calculating flexural strength is:

$$\sigma = \frac{3FL}{2bd^2}$$

and we are looking to see which toothpick can take in the maximum force, F. It directly follows that as we increase b, the width measured in mm, and d, the thickness measured in mm, the toothpick can withstand a greater force. Elementary analysis allows us to see that thickness is greater and the width is the same for the round toothpicks compared to the flat toothpicks; it is now straightforward to see the advantage of using the round hardwood toothpicks. the outcome of this game. It is essentially free points that even a defensive robot cannot stop. As of right now we are choosing to focus the endgame period, since lifting and raising the flag are relatively simple tasks.

We are considering purchasing the AndyMark field, as it would provide us a standardized field and a better replication of the interaction of the robot with the field during competition.

3 Software Architecture and Implementation

Detailed in this section is the method and reason to our software architecture. In particular, we aimed for our architecture to be stateless with respect to the current operation, to maximize efficiency of joystick checks (which are normally slow in RobotC), and to allow dynamic and easy reassignment of both controller information during teleop, as well as both movement sequencing and heuristic decision trees during autonomous.

We have chosen a particular structure for our code. We have rewritten `JoystickDriver.c` to better express our needs. By removing superfluous tasks from the driver, we maintained functionality while increasing our efficiency by approximately threefold. We have created a set of utility headers: `teleoputils.h`, `autoutils.h`, and `sharedutils.h` to address the need for macros, `#define` statements, and standardized functions. Both `teleoputils.h` and `autoutils.h` import `sharedutils.h`, so that file is never explicitly included in the main code body. Drivers were pulled from HiTechnic's online library for 3rd party sensors.

3.1 Teleoperated Mode

Teleoperated mode has the following requirements:

1. Smooth arcade driving
2. Easy reassignment of buttons
3. Stateless control of motors and robot state
4. Efficient button control and loop checking
5. **Must** use only one controller

In order to implement this effectively, we have implemented several macros. These macros allow us to later set the powers of the motors without much effort.

3.1.1 Drive Code and Reasoning

Our main body of code is run through the following:

```
task main() {
        while(true) {
                getJoystickSettings(joystick);
                checkJoystickButtons();
                setLeftMotors(powscl(JOY_Y1)-powscl(JOY_X1)/1.75);
                setRightMotors(powscl(JOY_Y1)+powscl(JOY_X1)/1.75);
        }
}
```

First, grab the current joystick configuration from the controllers. Then, check to see if any buttons have changed (`checkJoystickButtons();`). Then, set the motor powers for arcade drive.

Power scaling The `powscl(int)` function's definition is intended to compensate for the large deadband range which occurs under standard drive conditions. The controller's user really only needs two ranges: a high-precision, low power range near zero, and a low-precision, high power range near the maxima. While an exponential function could be used, it is much slower, and much more hard to tune. Instead, we draw two lines: a shallow slope for the first segment, then a large slope for the second segment of the controller. This provides both high precision and high power where needed. As the driver does not generally use the range in $[60, 85]\%$, there is no concern about the nonlinearity. The function is defined as follows:

```
float powscl(int xz) {
        float sign = (float)sgn(xz);
        float x = abs(xz)/128.0;
        if(x < DISTA)
                return 100* sign * (x*SLOPE);
        else
                return 100* sign * ((DISTA*SLOPE*(x-1.0) - x + DISTA) / (DISTA -
                    1.0));
}
```

Compensation for the Old and New Joystick Configuration It is necessary to compensate for both the old and new controller configurations. As the controller has been updated, the buttons have changed - however, competition rules premit the use of both controllers. Therefore, we must be able to accomodate this change if necessary. We have done so through the use of a define statement: if we have `#define ALTLOG`, then we switch to the old button layout.

Button Press Checking Requires a Stateless Organization In order to easily and effectively change the functionality of the controller, a particular design was implemented:

```
void invokeButton(int button, bool pressed) {
        switch(button) {
                case JOY_X:  if(pressed) {servo[servoL1] = 156; servo[servoL2] =
                    26;} else {} break;
                case JOY_Y:  if(pressed) {servo[servoL1] = 120; servo[servoL2] =
                    40;} else {} break;
                case JOY_A:  if(pressed) {} else {} break;
                case JOY_B:  if(pressed) {motor[mSpin] = 100;} else {motor[mSpin]
                    = 0;} break;
                case JOY_RB: if(pressed) {setArmMotors(100);}  else
                    {setArmMotors(0);} break;
                case JOY_LB: if(pressed) {setArmMotors(-100);} else
                    {setArmMotors(0);} break;
                case JOY_R3: if(pressed) {} else {} break;
                case JOY_L3: if(pressed) {} else {} break;
        }
}

bool t[8];
void checkJoystickButtons() {
        for(int i = 0; i < 8; i++) {
                if(joy1Btn(i) != t[i]) {
                        invokeButton(i, !t[i]);
                        t[i] = joy1Btn(i);
                }
        }
}
```

This may appear confusing at first, however, there are a couple points: `checkJoystickButtons()` is actually called from the main loop. It simply checks to see what buttons have changed on the controller, and calls the appropriate `invokeButton(int, bool)` arguments. In doing so, we can determine exact behavior on button presses with ease. As our robot is very simple, we do not need more than a handful of buttons, so most of them remain unassigned.

3.2 Autonomous Mode

3.3 Algorithms and Cartesian Mathematics

3.3.1 Hybrid Localization Using Gyroscopes & Odometers

Odometric Data We begin by assigning the following constants:

$$D_{ot} = \frac{\text{Distance}}{\text{odometer tick}} = \pi(\text{wheel diameter})/(\text{ticks/revolution})$$

$$\theta_{ot} = \frac{\theta}{\text{odometer tick}} = \pi\left(\frac{\text{wheel diameter}}{\text{distance between wheels}}\right)/(\text{ticks/revolution})$$

We can calculate $(x_{\text{enc}}, y_{\text{enc}}, \theta_{\text{enc}})$ from the odometer as follows:

$$\begin{aligned}
\mathrm{d}l &= l^t_{\text{enc}} - l^{t-1}_{\text{enc}} \\
\mathrm{d}r &= r^t_{\text{enc}} - r^{t-1}_{\text{enc}} \\
\mathrm{d}D &= \frac{1}{2}(\mathrm{d}l + \mathrm{d}r)D_{ot} \\
\mathrm{d}x_{\text{enc}} &= \mathrm{d}D\cos(\theta^t_{enc}) \\
\mathrm{d}y_{\text{enc}} &= \mathrm{d}D\sin(\theta^t_{enc}) \\
\mathrm{d}\theta_{\text{enc}} &= (\mathrm{d}r - \mathrm{d}l)\theta_{ot} \\
x_{\text{enc}} &= x^{t-1}_{\text{enc}} + \mathrm{d}x_{\text{enc}} \\
y_{\text{enc}} &= y^{t-1}_{\text{enc}} + \mathrm{d}y\text{enc} \\
\theta_{\text{enc}} &= \theta^{t-1}_{\text{enc}} + \mathrm{d}\theta_{\text{enc}}
\end{aligned}$$

l denotes the left side of the robot, and r denotes the right side of the robot. D denotes the distance.

Localization Algorithm The robots motor controller calculates position and orientation $(x_{\text{enc}}, y_{\text{enc}}, \theta_{\text{enc}})$ from encoder ticks and sends the data to an on-board computer. The mounted gyroscope communicates with a gyro driver which integrates the rate values into an absolute angle (θ_{gyro}). Global position $(x_{\text{rbt}}, y_{\text{rbt}})$ is found by transforming the translation vector from encoder space to gyroscope space. Global angle (θ_{rbt}) is the gyro angle (θ_{gyro}). The following describes the computation as an iterative algorithm:

$$\begin{aligned}
\mathrm{d}x &= x^t_{\text{enc}} - x^{t-1}_{\text{enc}} \\
\mathrm{d}y &= y^t_{\text{enc}} - y^{t-1}_{\text{enc}} \\
\mathrm{d}\theta &= \theta^t_{\text{gyro}} - \theta^t_{\text{enc}} \\
x^t_{\text{rbt}} &= x^{t-1}_{\text{rbt}} + \cos(\mathrm{d}\theta)\mathrm{d}x - \sin(\mathrm{d}\theta)\mathrm{d}y \\
y^t_{\text{rbt}} &= y^{t-1}_{\text{rbt}} + \sin(\mathrm{d}\theta)\mathrm{d}x + \cos(\mathrm{d}\theta)\mathrm{d}y \\
\theta^t_{\text{rbt}} &= \theta^t_{\text{gyro}}
\end{aligned}$$