# Engineering Notebook

## Team Administration

### Rubric Summary

#### Robot Rubric

<table>
<thead>
<tr>
<th>Topic</th>
<th>VHS Knights</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating Viable Solutions to the stated challenge</td>
<td>We have a large scoop to quickly pick-up a whole set of balls. Then we have a large bucket that the scoop can dump into. The bucket then raises to score into the high goal. We can use our bucket to hang on the bar. We can score around 78 points in both autonomous and driver control. We use rubber bands to help increase torque for both our scoop and bucket. We also used rubber bands to create a flat intake of our scoop.</td>
</tr>
<tr>
<td>Simple and/or complex systems</td>
<td>SPIRIT has a simple design of a scoop and bucket. The scoop simply picks up balls and dumps to the bucket. The bucket is raised and dumped in the high goal. The bucket lift also doubles in a lift for hanging on the bar.</td>
</tr>
<tr>
<td>Design process [documented in idea book or Engineering Notebook]</td>
<td>We used the “Think, Do, Test” system that was taught in the Vex IQ Curriculum. We took pictures before and after to document the changes that were made.</td>
</tr>
<tr>
<td>Utilization of resources [materials and parts information and instructions, people, and time]</td>
<td>We set goals at the beginning of each meeting. We also defined a meeting format to make sure we stayed within our time limits. We wanted to minimize the parts we use but build a sturdy, reliable robot.</td>
</tr>
<tr>
<td>Programming [Autonomous and/or tele-operated]</td>
<td>We programmed our robot to have both autonomous and driver control programs. We took common code between the two programs and made an include file so we didn’t have the same code in two different files.</td>
</tr>
<tr>
<td>Control systems</td>
<td>We programmed a Proportional line follower using our color light sensors and use a Touch LED sensor to transition between initialization and the start of the match.</td>
</tr>
<tr>
<td>Electrical systems</td>
<td>We added a battery voltage check at the beginning of each of our programs. We added the brain so that it can pivot up so that the battery can quickly be changed. We made sure our wires were safely threaded through the robot so that they wouldn’t get tangled or pinched.</td>
</tr>
<tr>
<td>Mechanical systems</td>
<td>We used a 15 to 1 gear ratio with 2 motors on each of our lifts. We added rubber bands to give additional torque. Our bucket uses a six bar design. We used sprockets and chain to make a 4 wheel drive.</td>
</tr>
<tr>
<td>Communication [written, electronic and/or oral as defined by the teacher]</td>
<td>We used white boards to share our ideas between each other and used Word to document our design changes. We made picture boards for our presentation.</td>
</tr>
<tr>
<td>Teamwork</td>
<td>We chose roles at the beginning of the season so we could all focus on a main task.</td>
</tr>
<tr>
<td>Creativity</td>
<td>SPIRIT can pick-up a whole set of balls in one scoop and can dump into the high goal.</td>
</tr>
</tbody>
</table>
## Project Rubric

<table>
<thead>
<tr>
<th>Topic</th>
<th>VHS Knights</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifies the topic of interest</td>
<td>Robotics is revolutionizing the farming industry.</td>
</tr>
<tr>
<td>Describe how math applies to robotics in the topic of interest</td>
<td>Math is critical to safely operate the “Driverless” tractor by converting the sensor data to information that can drive decisions.</td>
</tr>
<tr>
<td>Research from reliable sources</td>
<td>We used “Farm Show” Newspaper and Book and many websites. We also had a conference call with our expert, Mr. Terry Anderson, who has over 59 years of experience of automating manual tasks.</td>
</tr>
<tr>
<td>Research demonstrates well organized thought process</td>
<td>We researched how sensors worked and looked for how robotics is being used in farming.</td>
</tr>
<tr>
<td>Effective, creative presentation of findings</td>
<td>Maria dressed as the farmer and Isaac was SPIRIT, the driverless tractor. We compared our findings from SPIRIT to our robot.</td>
</tr>
<tr>
<td>Student understanding is present</td>
<td>Please come to our table to ask questions.</td>
</tr>
</tbody>
</table>
# Engineering Notebook

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rubric Summary</td>
<td>1</td>
</tr>
<tr>
<td>Robot Rubric</td>
<td>1</td>
</tr>
<tr>
<td>Project Rubric</td>
<td>2</td>
</tr>
<tr>
<td>Administration</td>
<td>1</td>
</tr>
<tr>
<td>Team Info</td>
<td>1</td>
</tr>
<tr>
<td>Team Photos</td>
<td>1</td>
</tr>
<tr>
<td>Team Profile</td>
<td>3</td>
</tr>
<tr>
<td>Team Administration</td>
<td>5</td>
</tr>
<tr>
<td>Team Sponsors</td>
<td>6</td>
</tr>
<tr>
<td>Our Season</td>
<td>6</td>
</tr>
<tr>
<td>Engineering Design Notes</td>
<td>1</td>
</tr>
<tr>
<td>Robot Version 1</td>
<td>1</td>
</tr>
<tr>
<td>Robot Version 2</td>
<td>10</td>
</tr>
<tr>
<td>Robot Version 3</td>
<td>18</td>
</tr>
<tr>
<td>Robot Version 4</td>
<td>27</td>
</tr>
<tr>
<td>State Competition</td>
<td>31</td>
</tr>
<tr>
<td>Robot Version 5</td>
<td>32</td>
</tr>
<tr>
<td>Robot Version 6</td>
<td>35</td>
</tr>
<tr>
<td>Programming</td>
<td>1</td>
</tr>
<tr>
<td>Summary</td>
<td>1</td>
</tr>
<tr>
<td>Program Structure</td>
<td>1</td>
</tr>
<tr>
<td>Autonomous Functions</td>
<td>3</td>
</tr>
<tr>
<td>Driver Control Functions</td>
<td>4</td>
</tr>
<tr>
<td>Team Work Functions</td>
<td>4</td>
</tr>
<tr>
<td>Restart Functions</td>
<td>4</td>
</tr>
<tr>
<td>Common Driver Functions</td>
<td>4</td>
</tr>
<tr>
<td>Common Functions</td>
<td>5</td>
</tr>
<tr>
<td>Project Design Notes</td>
<td>1</td>
</tr>
</tbody>
</table>

Contents - 2
Engineering Notebook

Rubric Summary ........................................ Project - 1
Presentation .................................................. Project - 1
Script ............................................................ Project - 1
Visual Aids .................................................... Project - 3
Sources of Information ...................................... Project - 8
Expert ............................................................ Project - 8
Books and Newspapers ..................................... Project - 8
Websites ........................................................ Project - 8
Administration

Team Info

Our team, the VHS Knights, is from Greenville, Wisconsin. Our team consists of four friends from three families. Team members are Maria [ ], Ben and Isaac [ ], and Michael [ ]. The families belong to a Christian home school group called the Valley Home Schoolers (VHS). Our team is coached by Joe and Mary Uchytil, and Fern School. Our head coach, Joe, is also head coach for VEX team, VHS Crusaders, and is a mentor for the VEX IQ team, VHS Electro Heads. The VHS Crusaders have also mentored our team.

Early in our season we chose the name VHS Knights and designed the logo for our t-shirts. After the Wisconsin state competition, we put together two videos. One video was a thank you and status update to our team project expert. The second video was a team promotional video.

Our team promoted the VEX IQ program at the Wisconsin State VEX Championship. We spent the day demonstrating our robot to individuals and put on an exhibition in front of the whole crowd. We were notified the next day that someone was planning to start 3 new teams.

Our team has enjoyed preforming community service projects. We participated at the Help for Homeless personal care drive, Christmas Cares Shoebox drive, and Walk for Wishes.

Our team loves California. We look forward to competing and sharing with kids from around the world.

Team Photos
Helping at Help for the Homeless personal care products drive.

Walk and Run for Wishes
Michael is our Software Leader. He is in 8th grade. He programmed the autonomous robot round that gets around sixty points.

Isaac is our Project Leader. He is in 7th grade. He found the article about the Driverless Tractor in the Farm Show newspaper.
Engineering Notebook
Team Administration

Maria is our Engineering Notebook Leader. She is in 6th grade. She came up with the skit and update the notebook.

Ben is our Hardware Leader. He is in 4th grade. He helped build the robot.
Team Administration

- Team name requirements
  - Unique
  - Reprehensive
  - Fun
  - Proud
  - Meaningful

- Team name ideas
  - VHS Knights
  - Super Stars
  - Magnetic Toads

- Roles
  - Hardware, Ben
  - Software, Michael
  - Project, Isaac
  - Notebook, Maria

- Meeting format
  - Prayer
  - Homework review
  - Goal setting
  - Work
  - Update Engineering notebook
  - Review work done
  - Homework
  - Teamwork games (if any)

- Team Name = VHS Knights
- Robot Name = SPIRIT
Team Sponsors
Our sponsors were the Kimberly-Clark Foundation, Skyline Technologies, Autonomous Tractor Corporation, Valley Home Schoolers Milwaukee School of Engineering, AZCO Inc., and our proud parents and friends. Kimberly-Clark is a global consumer product manufacturer. Skyline Technologies is a high level IT integration service provider headquartered in Appleton, WI. Autonomous Tractor Corporation not only gave us a financial donation, but also donated their time and expertise in the development of our project presentation. Autonomous Tractor Corporation has developed an autonomous tractor that is just being introduced on the market. Milwaukee School of Engineering is well known college for engineering. AZCO Inc. is a national industrial construction company. Our proud parents and friends gave us financial donations, but most important of all, gave us the encouragement to stretch our minds and our abilities. We have been greatly blessed by everyone’s support.

Our team also sold Vande Walle candy bars to help cover our team’s costs.

We look forward to proudly represent everyone who contributed to our success.

Our Season
We started bi-weekly meetings in August with another team, the VHS Electo Heads. Together we learned how to make the Vex IQ parts work and started building our first robots. At the end of each month, we held mini-competitions to help us understand how we could improve our robots. Through our regular Think-Do-Test cycles, the robots continued to improve. The team documented our robot changes and team activities for each meeting in our engineering notebook.

For our project, we brainstormed ideas and selected the best idea through voting. Through a farm magazine, we found a company that has developed an automated farm tractor. We interviewed the tractor inventor/company owner during an hour-long teleconference call. We integrated what we learned into our project presentation.

During the Wisconsin State competition, we won first place in the Robot Skills and Programming Skills challenges, the Teamwork challenge and the Excellence award. Our Programming Skills score of 51 and our Robot Skills score of 57 ranked us 1st place and 12th place respectively in the WORLD.

Since the state competition, we have explored numerous major changes to the robot. We ultimately returned to our original design and made incremental changes to that. We are looking forward to competing at the World Championship and are hoping that we can make Wisconsin and Valley Home Schoolers proud!
Built clawbot and began to learning how to program it.
Built new robot base that was larger and sturdier than the clawbot.

Top left – top side view. Used 2 omni wheels and the large wheel driving fast.

Top right – side view. Learned how to use twist on collars with wheels and shafts.

Bottom left – front view. Sturdy build to hold up brain and keep robot together.

Bottom right – top view. Secure uprights that can be used for adding lifting mechanisms.

Important Hardware Design Concepts learned:

- Beam = washer, collar, washer – useful to secure shafts between beams, especially for securing the wheels
- Use spacer and washer to reduce friction
- Brace corners
- Center of gravity above drive wheels
- Symmetry where possible
Goals

- Claw attached
- Update Engineering notebook
- All motors

Homework

- All- read rules

<table>
<thead>
<tr>
<th></th>
<th>Complexity</th>
<th>How many balls</th>
<th>Big ball</th>
<th>Design Picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maria</td>
<td>B</td>
<td>4</td>
<td>Maybe</td>
<td>Michael's idea was kinda like Maria's except there's another belt going up and it connects at their pivot points</td>
</tr>
<tr>
<td>Michael</td>
<td>C</td>
<td>7</td>
<td>Maybe</td>
<td>Maria's idea is a convare belt That sucked up balls and then would shoot them out into the goals</td>
</tr>
<tr>
<td>Isaac</td>
<td>A</td>
<td>5</td>
<td>Yes</td>
<td>Isaac's idea was a picker in the front of the robot and it would flip and drop a Bucky ball into a basket on the back.</td>
</tr>
<tr>
<td>Ben</td>
<td>C-</td>
<td>6</td>
<td>Maybe</td>
<td>Ben's idea was to change our whole robot design, having a straight up collection tube.</td>
</tr>
</tbody>
</table>
Worked on making claw picker and modifying robot base to hold the picker.
Increased to larger size picker that could pick up 3 balls at a time. Worked on attaching picker to robot base.

Lift with 2 bar design so that the picker would go straight up and down.

Picker from top.
Side view, front view, side view, back view, bottom view.
Goals

- Pick up ball

Worked on the claw and by the end of the night we got the claw to pick up a ball.

Side view, front view, top view. Started trying to program it.
Isaac prayed. Coach Joe gave class about torque the formula for finding the torque we’re putting on the motor is.

We played with some different ideas for picking up the balls.
Continued to work on forklift idea for picking up balls.
Robot Version 2
11/8/13

After mini-competition on Monday, we found that the claw picker was not a great way to pick up the balls. We re-evaluated and decided we needed to find a different, faster way to pick up the balls.

We calculated that there is a max of 156 points.

Requirements
- Under Bar

Priority
1. Small balls in high goal
2. Big balls in high goal
3. Small balls in rings
4. Hang on bar
Claw picker attachment.

Fork lift idea – top view and side view.

Conveyor belt – top view.
Engineering Notebook
Engineering Design Notes

11/11/13

We changed our vertical support beams from 2 by 16 beam, to 4 by 12. Today we made our conveyor belt so it could fit big balls and small balls.

Base with added up rights – top view, back view, side view and with base of conveyor belt added.
We built conveyor belt and added it to our conveyor base.

New Conveyor bot with conveyor belt added on top – side view, top view, back view of gearing needed to lift the conveyor attachment and front view with balls being taken in. We needed to use a 25:1 gearing ratio to get good torque to lift the conveyor attachment as it is heavy.
Conveyor bot conveyor being test driven by Isaac for first time, conveyor belt in action, gears running conveyor belt, and motor mount on conveyor belt.
We measured the height of big ball and small ball and found that the small ball height is 3in and big balls, 5in.

Idea on how to get conveyor belt to expand for big ball – there would be a sliding mechanism that when the big ball was being pulled in, the conveyor would slide upwards.

Another idea on how to get conveyor belt to expand for big ball – there would be pivot points so that the conveyor could swing up when the big ball was pulled in.

Conveyor was modified to have a larger intake gear and a smaller gear at the base so the robot could fit under the bar. Conveyor gearing and motor were moved to the intake side of conveyor.
11/22/13

We looked at different ideas on how to have a bucket in the back end to fill with intake conveyor and then dump from the bucket into the goals.

Idea 1 – an elevator lift, Idea 2 – a scissors lift, Idea 3 – 4 bar lift

Team agreed to try Idea 3 – the 4 bar lift.

Isaac built first prototype of 4 bar lift. Decided that there as a lot of wasted space and needed to be wider and more compact to fit on the back end of the robot. This one could only handle 2 small balls.
We worked on another bucket attached with a 4 bar lift.

These lifts were wider, but there was still too much wasted space over the back end of the robot.
11/25/13

We worked on a new bucket for the back of the robot.

12/2/13

We prepared our robot for our mini-competition.

Robot Version 3
12/6/13

We learned: how to make a six bar lift; how to use pulleys; and what sprockets are used for with chain.

Think: How to raise the bucket

Do: Built a six bar for our bucket; we also added 2 - 2x20 vertical supports that we can add the 6 bar attachment to the base of the robot

Test: the six bar worked but the mechanism that connected the bucket to the six bar didn't really work. We also found the 2x20 is too tall to make it under the bar.
Think: Tires spun during our mini-competition

Do: We added chain and sprockets to make a 4 wheel drive.

Test: We are still seeing some spinning while driving.
We split into three groups: Michael doing Software, Isaac working on our bucket, and Ben and Maria working on our logo.

We learned:

Think: We need to attach our bucket to the robot.

Do: We attached our bucket and 6 bar lift to robot. We geared the lift to be a 25/1 gear reduction.

Test: Lift worked well, but we have some rubbing areas that need to be fixed. We also have a weak spot in using the shaft bushing.
Ben and Maria made our logo, Michael worked on our programming, and Isaac worked on our scoop.

**12/20/13**

Isaac worked on the six bar. Ben worked on gears. And Maria worked on the basket. Michael worked on the putting the scoop down, moving forward, and turning, he work on the six part strategy: one, get group one, two, get group 2, three, dump the balls, four, get group three, five, get group four, six, dump the balls.

The Gearing for the scoop. Used a 15/1 gear reduction. Found that we need to increase the gear reduction even more for the size load we are looking to pick up.
Engineering Notebook
Engineering Design Notes

12/23/13

Michael worked programming the initialization of putting the scoop down. Isaac worked on six bar lift. Ben and Maria worked on the basket.

Basket

Six bar for bucket.
Isaac and Ben came over to work on the robot. Isaac worked on the basket. Ben worked on the scoop. Maria worked on the gears.
Gears – changed from 15/1 to 25/1.
Engineering Notebook
Engineering Design Notes

1/1/14

Maria worked on the bucket.

Bucket:

Side, front.

Side, back
1/3/14

Ben, Isaac, and Maria worked tweaking the robot.

Bucket

Scoop
Robot Version 4
1/6/14

We had our mini competition. We brainstormed about our project topics:

1. Medical & surgery
2. Manufacturing
3. Farming
4. Education

We chose to study Farming.

1/10/14

We saw that touch LED sensor was in a bad space, because it would run over the user’s hand.

Before  

After

We saw the lift’s motor was too slow, so we decided that we could take the rear drive motors off to give the lifts more power so we could change the gearing to increase the speed of the lifts.

Rear Motors removed.

Action items for next meeting:

1. Change to 4 wheel drive with just 2 motors by adding chain drive to the outside of the wheels.
2. Modify the lift gearing to add second motor for both the scoop and bucket and lower the gear reduction to get more speed.
3. Look at touch LED to see if location will be affected by the chain drive above.

We researched on how robots are used in farming.

1/13/14

Michael worked on programming and Isaac, Ben and Maria worked on the project. We decided to do our project on robot natation
Ben ad Isaac came over. They and Maria worked on the robot gears and added an extra motor to the Scoop and the bucket.

Before

After

Design Notes - 28
1/17/14

We made some hooks to hang on the bar

Hanging on the bar

1/20/14

Ben, Isaac, and Maria worked on project, while Michael worked on software.

1/22/14

Ben and Isaac came over and worked the project with Maria.

1/23/14

Maria worked on the coding the limits for the scoop. They also learned that if the lifts go weird and quit working it's because of the battery is low. We need to add a battery check to our programs.
Engineering Notebook
Engineering Design Notes

1/24/14

Goals:
1. Decide skit roles: Team member 1- Michael, Team member 2- Ben, Farmer- Maria, and SPIRIT- Isaac.
2. Decide driving pairs and positions: Isaac and Maria, and Michael and Ben.
3. Finalize skit
4. Drafts visual aides
5. Brainstorm costumes
6. Time line- paper chains – Isaac – Monday

1/27/14

Goals:
1. Practice driving – high scores in the 40s
2. Practice skit – check
3. Review Visual aids – needs work on these
4. Practice questions – homework
5. Review Rubrics – Robot and Project
6. Name Robot

1/29/14

Goals:
1. Name Robot
   a. Suggested Names include
      i. Shadow
      ii. SPIRIT
      iii. Einstein
      iv. King Arthur
      v. Sir Lancelot
      vi. Excalibur
      vii. Ghez Hokon
2. Practice skit
3. Options for wagon – use red wagon
4. Visual Aids – printed and taped to boards
5. Review costume or SPIRIT – will use shirt or jacket instead of box
6. Update Notebook
Goals

1. Name robot – SPIRIT
2. Practice robot – scored in the 50s. High score was 57.
3. Practice skit – videotaped last presentation to review how we looked.

2/3/14

Goals

- Practice driving
- Practice skit
- Mini competition
- Take team pics

2/7/14

We practiced driving and skit.

State Competition

2/8/14

Our State competition!

We took 1st place, won the excellent award, Teamwork, Driver and Programming Skills Awards! Our score of 51 for the programming skills is #1 in the world and our score of 59 for the driver’s skills is #12 in the world!

The Team and Awards
Engineering Notebook
Engineering Design Notes

2/17/14

Goals

- Think up ideas to do in CA
- Think up ideas about Robot changes
- Fundraising-
  - Corporate sponsors
    - Gulfstream
    - UPS
    - KC
    - John Deer
    - Autonomous Tractor corp.
    - Farm Show
  - Robot Driving in mall
  - Selling candy
  - Silent auction
  - Culvers

Robot Version 5
2/24/14

We talked about fundraising, and the videos, how to make the videos, recording, and editing.

We took the scoop off so we could experiment ideas to gather balls faster.

SPIRIT before we took the scoop off.
Engineering Notebook
Engineering Design Notes

2/28/14

We edited the promotional video and thought up problems with the bucket

- Dump lever hits wall when turning
- Too small
- Balls can fall out when dumping

We brainstormed ideas to try

- Take off the dump mechanism
- Make bucket bigger
- Change how to dump bucket
- Tilt bucket so that gravity grabs the balls and pull the back.

3/3/14

We edited the promotional video. And we worked on a conveyer belt intake.

Isaac’s idea was to have a conveyer like the our earlier idea but put the balls into our bucket like our scoop and bucket but no scoop

Maria’s idea was a robot with spinners like our VEX team
Engineering Notebook
Engineering Design Notes

3/7/14

Here is a comparison chart

<table>
<thead>
<tr>
<th>SPIRIT with conveyer intake</th>
<th>Spinner intake (Sir Lancelot)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="" /></td>
<td><img src="image2.png" alt="" /></td>
</tr>
<tr>
<td>Could hold lots of balls</td>
<td>Can't hold a lot of balls</td>
</tr>
<tr>
<td>Hard to drive around</td>
<td>Easy to maneuver</td>
</tr>
<tr>
<td>Slow intake</td>
<td>Fast intake</td>
</tr>
<tr>
<td>Can't handle the big ball</td>
<td>Easy to pick up the big ball</td>
</tr>
</tbody>
</table>

We decided to give Sir Lancelot a try

3/8/14

Isaac came over and helped start making SPIRIT to be more like Sir Lancelot.
3/10/14

Michael worked on intake, Maria worked on updating the notebook, and Isaac and Ben worked on gearing for the lift and we converted our 6 bar to 8 bar to get the extra height. We attached the brain securely to the back of the robot.

Robot Version 6
3/14/14

We found that Sir Lancelot’s intake design made the balls go flying when he tried to take them in. This made it unpredictable for autonomous programming. We felt it would be better to go back to our original SPIRIT design and make improvements to that design.

We discussed how we could get more points during the match by going after 6 middle balls before getting the two sets of balls at the end of the table.

3/15/14

Isaac and Michael came over to help Maria with the changing the base back to SPIRIT’s original design. We made improvements on things we learned from our Sir Lancelot’s design.
3/17/14

Isaac worked on the scoop, Michael worked on attaching the bucket, and Maria updated the notebook.

Scoop – made longer with angled sides

Bucket – added temporary spacers until we can widen the bucket
Engineering Notebook
Engineering Design Notes

3/19/14

Ben and Isaac came over to work on the robot. Isaac worked on programming, and Maria worked on the notebook. We rewired the robot and test drove SPIRIT. We found the scoop to be too big and that it pushed balls all over. Brainstormed some ideas to use rubber bands in the bottom of the scoop.

3/21/14, 3/22/14, and 3/23/14

Michael and Maria. Michael worked on widen and lengthen the bucket. And Maria worked on building a new scoop.

The bucket was made longer, wider, removed the dumping mechanism, the back was made flat, added gears to dump, and added a backstop to keep the balls in until we were ready to dump.

Side  Back

Gears

Design Notes - 37
Engineering Notebook
Engineering Design Notes

We change the Bucket Lift form a 6 bar to 8 bar to get more height.

We redesigned the scoop

3/24/14

Maria worked on updating the notebook. Michael, Ben, and Isaac worked on programming the robot and attached the scoop. Michael thought up ideas for initialization. We saw that the rubber bands where rubbing on the floor, Ben and Isaac thought up ideas to make the back rubber bands to be a little bit taller.
3/28/14

Maria worked on the side rubber bands on the scoop and upgraded then it to what we call ‘Maria’s Web.’ Michael worked on attaching the gyro and light sensor. Isaac worked on dumping mechanism. Ben worked on putting sides on the bucket.

3/29/14

Isaac came over and helped work on the robot. He worked on the dumping mechanism. Maria worked on ‘Maria’s Web’ to make it sturdier. Later Michael came and worked on programming.

Dumping mechanism – as the 6 bar goes up, the gears attached to the 6 bar cause the gear attached to the back gate to turn downward. By the time the bucket is raised to the height of the high goal, the back gate lays flat and pushes the bucket to swivel downward and dump them into the goal.

Side view

Maria’s Web on the scoop

3/31/14

Isaac, Ben and Maria worked on the skit, and Michael worked on programming.
Engineering Notebook
Engineering Design Notes

4/2/14

Maria and Ben changed the 5 to 1 gearing to about an 8 to 1 and to back to a 15 to 1 gearing. Isaac, Ben and Maria did test run of the robot. Michael worked on programming.

<table>
<thead>
<tr>
<th>5 to 1 gearing</th>
<th>25 to 3 gearing</th>
<th>15 to 1 gearing</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.jpg" alt="5 to 1 gearing" /></td>
<td><img src="image2.jpg" alt="25 to 3 gearing" /></td>
<td><img src="image3.jpg" alt="15 to 1 gearing" /></td>
</tr>
</tbody>
</table>

We also rebuilt our scoop because the motors still could not lift the scoop at the 15:1 ratio. We moved the pivot point closer the gears.

New scoop

![New scoop](image4.jpg)
We had been noticing that the robot was not able to move as well as it used to. The robot seemed to be "sluggish". We found that the RobotC programming update had switched the motor power function that only set our power to half of what they used to. That left us with not being able to lift our scoop like we used to be able to. We need to convert all our programs to use the new function.

Team decided to do a night of driving the robot. In the first round we repeated the think, do, test process the problem: when the robot try's to dump the balls spill. So Michael set to work programing a button to get it to a good dumping height. It looks good so far.

We enjoyed a night of team building activities together.

Michael spent all day programming. He found a couple of hardware issues. He moved the rubber bands on the scoop back to running on the floor.

Michael spent most of the day programming and Isaac spent a little time. They got the autonomous nearly done. Hardware issues that still need to be corrected: webbing on the scoop needs to be improved.
We practiced the robot and skit. Maria made Maria's Wed better and stronger.

Before

After

4/9/14

We practiced skit and robot driving. Michael stayed over and worked on programming. Michael added a V configuration on the back of the robot to help with lining up the robot to dump into the high goal.
Maria spent time calculating what advantage the rubber bands provide. They provide extra force in lifting the bucket up. And with them around the 60 tooth gear set, they provide extra torque for the scoop.

<table>
<thead>
<tr>
<th>Radius of Gear</th>
<th>cm</th>
<th>2.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversion Factor</td>
<td>g cm / newton meter</td>
<td>10197.16213</td>
</tr>
<tr>
<td><strong>Force = weight created by Rubber Bands at Distance-cm</strong></td>
<td>Torque = Force(g) * Radius of Gear(2.8cm)</td>
<td></td>
</tr>
<tr>
<td>#Rubber Bands</td>
<td>Distance-cm</td>
<td>Force-g</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>560</td>
</tr>
<tr>
<td>1</td>
<td>19.5</td>
<td>690</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>980</td>
</tr>
<tr>
<td>2</td>
<td>19.5</td>
<td>1330</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>1390</td>
</tr>
<tr>
<td>3</td>
<td>19.5</td>
<td>2020</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>1590</td>
</tr>
<tr>
<td>4</td>
<td>19.5</td>
<td>2530</td>
</tr>
</tbody>
</table>

Force = g at what distance
Force * radius of gear = torque
max & min torque = torque at 19.5 & 13 cm

4/11/14
We practiced skit and robot driving.

4/12/14
We practiced robot driving while our parents met to finalize our trip plans. We also watched a movie together.

4/14/14
We practiced on skit and robot driving. We videotaped one of our skits and watched it.
4/15/14

We practiced the skit and robot driving. Afterward Michael finalized programming. Michael added a cancellation button that gets out of the super dumping program, fixed the FUp button which gets the bucket up to the right height to dump but does not dump, wrote an restart program if we have an I2C error, and lowered the set scoop position in Teamwork.

4/16/14 and 4/19/14

We practiced skit and robot driving.
Engineering Notebook
Programming Design Notes

Programming

Summary
We programmed our robot to have autonomous, driver control, and team work programs. We took common code between the programs and made two include file so we don't duplicate common code in our programs.

We use three color light sensors to follow and detect lines. We programmed a Proportional line follower. We also use a touch LED sensor to transition between initialization and the start of the match.

In our driver control and team work programs, we initialize our scoop and bucket. We then set minimum and maximum limits so our motors will not get over driven.

We put a battery check in both programs, so that SPIRIT does not fail in the match due to low voltage. We use the GoPro slow motion program to find out how to make SPIRIT better.

We coded the following buttons:

- E down button - to make a **SLOW** turn to line SPIRIT up to the high goal with ease
- Automatic buttons
  - E up and F down buttons - to automatically get the balls next to the goal
  - F down and E down
    - dumps the bucket full of balls into the goal
    - transfers the balls from the scoop to the bucket
    - lifts the bucket full of balls into the goal
    - turns towards the hanging bar
    - hangs on the hanging bar
  - E up - cancels out of the two automatic buttons

Program Structure

We have 4 main programs with two supporting libraries.

<table>
<thead>
<tr>
<th>Program Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knights-Auton.c</td>
<td>The autonomous program</td>
</tr>
<tr>
<td>Knights-Driver.c</td>
<td>The driver skills program</td>
</tr>
<tr>
<td>Knights-Team.c</td>
<td>The team work program</td>
</tr>
<tr>
<td>Knights-Restart.c</td>
<td>Emergency restart program during driver and team work challenge in case we get an I2C error during the match</td>
</tr>
<tr>
<td>CommonDriver.c</td>
<td>Contains code that is used in the driver, team work and restart programs</td>
</tr>
<tr>
<td>CommonCode.c</td>
<td>Contains code that is used in all the programs</td>
</tr>
</tbody>
</table>
Engineering Notebook
Programming Design Notes

Here's a diagram showing the relationships between the programs.

![Diagram showing relationships](image)

See Autonomous, Driver Control, Team Work, Common Code, and Common Driver tabs for more programming details.
Engineering Notebook
Programming Design Notes

**Autonomous Functions**

This code is only used during the autonomous competition.

<table>
<thead>
<tr>
<th>Function Names</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>task main()</td>
<td>• this is the main task that runs first</td>
</tr>
<tr>
<td></td>
<td>• calls InitializeAutonomous()</td>
</tr>
<tr>
<td></td>
<td>• calls RunAutonomous()</td>
</tr>
<tr>
<td>void InitializeAutonomous()</td>
<td>• check the battery voltage</td>
</tr>
<tr>
<td></td>
<td>• initialize the scoop</td>
</tr>
<tr>
<td></td>
<td>• calibrate light sensors</td>
</tr>
<tr>
<td></td>
<td>• initialize our position</td>
</tr>
<tr>
<td></td>
<td>• initialize the bucket</td>
</tr>
<tr>
<td>void RunAutonomous()</td>
<td>• calls GetGroup1()</td>
</tr>
<tr>
<td></td>
<td>• calls GetGroup2()</td>
</tr>
<tr>
<td></td>
<td>• calls GetGroup3()</td>
</tr>
<tr>
<td></td>
<td>• calls DumpAndHang()</td>
</tr>
<tr>
<td>void GetGroup1()</td>
<td>Get the six balls along the goal and get the three balls at the end of</td>
</tr>
<tr>
<td></td>
<td>the table</td>
</tr>
<tr>
<td>void GetGroup2()</td>
<td>Get the remaining balls at the end of the table and get two balls</td>
</tr>
<tr>
<td></td>
<td>from the side of the table</td>
</tr>
<tr>
<td>void GetGroup3()</td>
<td>Get the rest of the balls along the side of the table</td>
</tr>
<tr>
<td>void DumpAndHang()</td>
<td>• dumps the bucket full of balls into the goal</td>
</tr>
<tr>
<td></td>
<td>• transfers the balls from the scoop to the bucket</td>
</tr>
<tr>
<td></td>
<td>• dumps the bucket full of balls into the goal</td>
</tr>
<tr>
<td></td>
<td>• turns towards the hanging bar</td>
</tr>
<tr>
<td></td>
<td>• hangs on the hanging bar</td>
</tr>
</tbody>
</table>

![Diagram of the autonomous functions and the autonomous competition field with groups 1, 2, and 3 highlighted]
Driver Control Functions
This code runs during the driver control competition.

<table>
<thead>
<tr>
<th>Function Names</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>task main()</td>
<td>This is the main task that runs first</td>
</tr>
<tr>
<td>void InitializeDriverControl()</td>
<td>Initialize the different parts of the robot</td>
</tr>
</tbody>
</table>

Team Work Functions
This code runs during the team work competition.

<table>
<thead>
<tr>
<th>Function Names</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>task main()</td>
<td>This is the main task that runs first</td>
</tr>
<tr>
<td>void InitializeTeamControl()</td>
<td>Initialize the different parts of the robot</td>
</tr>
</tbody>
</table>

Restart Functions
This code runs during the team work or driver control competition.

<table>
<thead>
<tr>
<th>Function Names</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>task main()</td>
<td>This is the main task that runs first</td>
</tr>
<tr>
<td>void InitializeRestart()</td>
<td>Initialize the different parts of the robot</td>
</tr>
</tbody>
</table>

Common Driver Functions
This code by the driver, team work and restart programs

<table>
<thead>
<tr>
<th>Function Names</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>int Trim(int joystick)</td>
<td>Remove small joy stick values</td>
</tr>
<tr>
<td>void RunDriverControl()</td>
<td>Run driver control code</td>
</tr>
</tbody>
</table>
# Engineering Notebook

## Programming Design Notes

### Common Functions

This code is shared between the driver control and the autonomous.

<table>
<thead>
<tr>
<th>Function Names</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>void Write(string *text)</td>
<td>Write a message to the debug stream</td>
</tr>
<tr>
<td>void WaitUntilMotorMoving(tMotor motor)</td>
<td>Wait until a motor begins moving</td>
</tr>
<tr>
<td>bool IsBatteryOK()</td>
<td>Displays an alarm if the battery voltage is low</td>
</tr>
<tr>
<td>void Move2(bool moveForward, float millimeters, int speed, bool waitUntilDone)</td>
<td>Moves the robot forward and backwards and allows the move to time out</td>
</tr>
<tr>
<td>void ResetTimer()</td>
<td>Reset time and display time work together</td>
</tr>
<tr>
<td>void DisplayTimer()</td>
<td>Display the time since the reset timer</td>
</tr>
<tr>
<td>bool IsRobotMoving()</td>
<td>Check if robot is moving</td>
</tr>
<tr>
<td>void SetRobotSpeed(int speed)</td>
<td>Set the robot speed</td>
</tr>
<tr>
<td>void WaitUntilLineCrossed(tSensors lightSensor)</td>
<td>Wait until the crossing line is crossed</td>
</tr>
<tr>
<td>void MoveUntilLine(bool moveForward, int speed, tSensors lightSensor)</td>
<td>Move until we cross a line</td>
</tr>
<tr>
<td>void FollowLineUntilCrossingLine(int speed, tSensors lightSensor, bool rightSideOfLine)</td>
<td>Follow the line until the crossing line sensor senses a crossing line</td>
</tr>
<tr>
<td>void TurnOneWheel(bool turnLeft, int robotDegrees, int speed, bool forward)</td>
<td>Turn the robot using one wheel</td>
</tr>
<tr>
<td>void WaitUntilTouchAndRelease()</td>
<td>Wait until the touch LED is touched and released</td>
</tr>
<tr>
<td>void Move(bool moveForward, float millimeters, int speed)</td>
<td>Moving straight in millimeters</td>
</tr>
<tr>
<td>void SetScoopPosition(float scoopDegrees, int speed, bool waitUntilDone)</td>
<td>Set the scoop to a certain position</td>
</tr>
<tr>
<td>void SetBucketPosition(float bucketDegrees, int speed, bool waitUntilDone)</td>
<td>Set the bucket to a certain position</td>
</tr>
<tr>
<td>void WaitUntilScoopPosition(float scoopDegrees, bool movingUp)</td>
<td>Wait until the scoop reaches a position</td>
</tr>
<tr>
<td>void InitializeBucket()</td>
<td>Initialize the bucket</td>
</tr>
<tr>
<td>void InitializeScoop()</td>
<td>Initialize the scoop</td>
</tr>
<tr>
<td>void CalibrateLightSensors()</td>
<td>Calibrate light sensors</td>
</tr>
<tr>
<td>void CalibrateRobotPosition()</td>
<td>Calibrate robot position so we know where we are</td>
</tr>
<tr>
<td>void WaitUntilDistance(float millimeters)</td>
<td>Wait until the robot moves a distance</td>
</tr>
<tr>
<td>void Initialize()</td>
<td>Initialize the different parts of the robot</td>
</tr>
</tbody>
</table>
// Include the common code used in both autonomous and driver skill
#include "CommonCode.c"

// Initialize the autonomous
void InitializeAutonomous()
{
    // Line followers
    leftLineFollowerBlack = 0;
    leftLineFollowerWhite = 0;
    rightLineFollowerBlack = 0;
    rightLineFollowerWhite = 0;

    // Crossing line
    crossingLineBlack = 0;
    crossingLineWhite = 0;
    degreesPerMillimeter = 1.81;

    setMotorCurrentLimit(rightWheel, 512);
    setMotorCurrentLimit(leftWheel, 512);

    // Is the battery okay?
    if(IsBatteryOK() == FALSE)
    {
        // No. Wait to hear the sound
        waitMsec(3000);
    }

    // Initialize the scoop
    InitializeScoop(75);

    // Calibrate light sensors
    CalibrateLightSensors();

    // Move the robot over so we line up with the edge of the line
    TurnOneWheel(false, 30, 30, true);
    TurnOneWheel(true, 30, 30, true);

    // Move to the start of the line
    Move(false, 150, 30);

    // Position the robot on the table by following the line until the crossing
    FollowLineUntilCrossingLine(10, leftLineFollower, true);

    // Initialize the bucket
    InitializeBucket(15);

    // Move the robot over so we line up with the edge of the line
TurnOneWheel(true, 17, 10, false);
TurnOneWheel(false, 16, 10, false);

// Set the scoop down so it is in a legal position
SetScoopPosition(15, 30, false);

// Backup so the we are in a legal position
Move(false, 245, 30);
}

// Get the first group of balls
void GetGroup1()
{
    // Set the scoop up so we don't hit balls
    SetScoopPosition(30, 30, false);

    // Move the robot into a better position to get the balls
    TurnOneWheel(false, 80, 70, true);
    TurnOneWheel(true, 75, 70, true);

    // Put scoop down so we can get the balls
    SetScoopPosition(5, 70, false);

    // Move the bucket down so it doesn't hit the bar
    SetBucketPosition(0, 90, false);

    // Back up so we don't hit the balls when we set the scoop down
    Move(false, 75, 70);

    // Move forward until the front edge of the scoop is just past the goal
    Move(true, 125, 70);

    // Turn towards the goal so the scoop touches and runs along the goal
    TurnOneWheel(true, 5, 90, true);

    // Move along the goal
    Move(true, 825, 100);

    // Get in position to get the three balls at the end of the group
    TurnOneWheel(true, 15, 70, true);
    Move(true, 350, 70);
    TurnOneWheel(false, 20, 70, true);

    // Move forward to get the balls but time out when hitting the wall
    Move2(true, 200, 100, true);
}

// Get the second group of balls
void GetGroup2()
{
    // Start lifting the scoop
    SetScoopPosition(150, 100, false);

    // Back up so scoop doesn't hit the wall
    Move(false, 100, 30);

    // Wait until the scoop is all the way up
    WaitUntilScoopPosition(30, true);
// Turn one wheel so we can back up to put the scoop down
TurnOneWheel(false, 30, 70, true);

// Wait until the scoop is all the way up
WaitUntilScoopPosition(130, true);

// Put scoop down so we can get the balls
SetScoopPosition(5, 90, false);

// Move the bucket up so we don't hit the wall
SetBucketPosition(30, 50, false);

// Back up so we can get the balls
Move(false, 150, 30);

// Wait until the scoop is most of the way down
WaitUntilScoopPosition(20, false);

// Move forward to start getting the balls
Move(true, 112, 90);

// Turn the robot so it doesn't hit the wall
TurnOneWheel(false, 20, 95, true);

// Move forward to get the balls
Move(true, 100, 95);

// Turn the robot so it doesn't hit the wall
TurnOneWheel(false, 10, 95, true);

// Move forward to push the balls into the corner and time out
Move2(true, 350, 100, true);

// Turn the robot so it is square to the wall
TurnOneWheel(true, 10, 100, false);

// Move forward to push the balls into the scoop and time out
Move2(true, 150, 90, true);

// Set the bucket down so we can transfer balls to the bucket
SetBucketPosition(0, 90, false);

// Get the second group of balls
void GetGroup3()
{
    // Start lifting the scoop
    SetScoopPosition(140, 100, false);

    // Back up so scoop doesn't hit the wall
    Move(false, 350, 30);

    // Wait until the scoop is all the way up
    WaitUntilScoopPosition(130, true);

    // Turn so we can get the third group of balls
    TurnOneWheel(false, 25, 100, true);
// Bring the scoop down so we can pick up the balls
SetScoopPosition(8, 100, false);

// Wait until the scoop is most of the way down
WaitUntilScoopPosition(15, false);

// Move forward to get the balls
Move(true, 225, 100);

// Turn so we don't hit the wall
TurnOneWheel(false, 30, 100, true);

// Move forward to get the balls
Move(true, 100, 100);

// Turn so we don't hit the wall
TurnOneWheel(false, 15, 100, true);

// Move forward until it can't move any further
Move2(true, 175, 100, true);

// Back up slowly so we don't loose the balls
Move(false, 100, 40);

// Start lifting the scoop
SetScoopPosition(30, 100, true);

// Dump the balls and hang
void DumpAndHang()
{
    // Bring the bucket up so we can dump
    SetBucketPosition(73, 50, false);

    // Turn backwards so the robot can square itself with a line
    TurnOneWheel(false, 65, 90, false);

    // Square the robot with the line
    SquareToLine(false, 30);

    // Turn robot towards the corner of the goal
    TurnOneWheel(false, 45, 40, false);

    // Back up to the goal until we hit the goal
    Move2(false, 500, 100, true);

    // Lift the bucket until the balls dump out
    SetBucketPosition(100, 100, true);

    // Wait so the balls can exit the scoop
    waitIMsec(500);

    // Start the bucket going down so we can get the balls from the scoop
    SetBucketPosition(5, 100, false);

    // Move forward so the bucket doesn't hit the goal
    Move(true, 250, 90);
// Wait until the scoop is all the way up  
WaitUntilBucketPosition(35, false);

// Start lifting the scoop  
SetScoopPosition(140, 100, true);

// Start bringing the scoop down  
SetScoopPosition(5, 100, false);

// Start the bucket going down so we can get the balls from the scoop  
SetBucketPosition(73, 100, false);

// Wait until the scoop is most of the way up  
WaitUntilBucketPosition(65, true);

// Back up to the goal until we hit the goal  
Move2(false, 275, 90, true);

// Lift the bucket until the balls dump out  
SetBucketPosition(100, 100, true);

// Move forward to get into the turning position  
Move(true, 300, 90);

// Turn towards the hanging bar  
TurnOneWheel(true, 67, 80, false);  
TurnOneWheel(false, 67, 80, true);

// Move forward into the hanging bar and time out  
Move2(true, 600, 90, true);

// Bring the bucket down so we hang.  
SetBucketPosition(0, 100, true);

// Run the autonomous program  
void RunAutonomous()  
{
  displayTextLine(1," ");  
  displayTextLine(2," ");  
  displayTextLine(3," ");  
  displayTextLine(4," ");

  // Reset the timer so we can see how long the program runs  
  ResetTimer();

  // Get the first group of balls  
  GetGroup1();

  // Get the second group of balls  
  GetGroup2();

  // Get the third group of balls  
  GetGroup3();

  // Dump and hang on the bar
DumpAndHang();

// Display how long the program ran
DisplayTimer();

// Sleep so we can see the time on the display
sleep(30000);
}

// This is the main task that runs first

task main()
{
    // Initialize the main
    InitializeMain();

    // Initialize variables

    // Clear the debug stream and write a message
clearDebugStream();
    Write("Program Starting");

    // Make the LED display Red
    // This means we are waiting for the LED to be touched
    setTouchLEDRGB(touchLED, 255, 0, 0);
    setTouchLEDBlinkTime(touchLED, 0, 0);

    // Wait until user touches the LED touch sensor
    Write(""");
    Write("Pre-initialize mode");
    Write("Waiting for touch");
    WaitUntilTouchAndRelease();
    Write("Touched");

    // Make the LED display flashing Yellow
    // This means the robot is initializing
    setTouchLEDRGB(touchLED, 255, 255, 0);
    setLEDBlinkTime(touchLED, 10, 10);

    // Initialize the robot
    InitializeAutonomous();

    // Make the LED display solid Green
    // This means the robot is ready to run and waiting for user to touch the bc
    setLEDRGB(touchLED, 0, 255, 0);
    setLEDBlinkTime(touchLED, 0, 0);

    // Wait until user touches the LED touch sensor
    Write(""");
    Write("Ready mode");
    Write("Waiting for touch");
    WaitUntilTouchAndRelease();
    Write("Touched");

    // Make the LED display flashing Blue
    // This means the robot is running
    setLEDRGB(touchLED, 0, 0, 255);
    setLEDBlinkTime(touchLED, 10, 10);
File: C:\Data\RobotC Programs\Knights-Auton.c

    // Run autonomous
    RunAutonomous();

    // Turn the light off
    setTouchLEDRGB(touchLED, 0, 0, 0);
    Write("Program Done!");
}
```c
#include "CommonCode.c"
#include "CommonDriver.c"

void InitializeDriverControl()
{
    // Line followers
    leftLineFollowerBlack = 0;
    leftLineFollowerWhite = 0;
    rightLineFollowerBlack = 0;
    rightLineFollowerWhite = 0;

    // Crossing line
    crossingLineBlack = 0;
    crossingLineWhite = 0;
    degreesPerMillimeter = 1.81;

    // Is the battery okay?
    if(IsBatteryOK() == false)
    {
        // W0. Wait to hear the sound
        wait1Ms(3000);
    }

    // Initialize the bucket
    InitializeBucket(0);

    // Initialize the scoop
    InitializeScoop(55);

    // Calibrate light sensors
    CalibrateLightSensors();

    // Calibrate robot position
    CalibrateRobotPosition();

    // Set the bucket closer to the floor so it will be in a legal position
    SetBucketPosition(0, 30, false);

    // Set the scoop down so it is in a legal position
    SetScoopPosition(10, 30, true);
}

// This is the main task that runs first
```

Page 1 of 2
{
    // Initialize the main
    InitializeMain();

    // Clear the debug stream and write a message
    clearDebugStream();
    Write("Program Starting");

    // Make the LED display Red
    // This means we are waiting for the LED to be touched
    setTouchLEDRGB(touchLED, 255, 0, 0);
    setTouchLED BlinkTime(touchLED, 0, 0);

    // Wait until user touches the LED touch sensor
    Write(""");
    Write("Pre-initialize mode");
    Write("Waiting for touch");
    WaitUntilTouchAndRelease();
    Write("Touched");

    // Make the LED display flashing Yellow
    // This means the robot is initializing
    setTouchLED RGB(touchLED, 255, 255, 0);
    setTouchLED BlinkTime(touchLED, 10, 10);

    // Initialize the robot
    InitializeDriverControl();

    // Make the LED display flashing Blue
    // This means the robot is running
    setTouchLED RGB(touchLED, 0, 0, 255);
    setTouchLED BlinkTime(touchLED, 10, 10);

    // Run the program
    RunDriverControl();

    // Turn the light off
    setTouchLED RGB(touchLED, 0, 0, 0);
    Write("Program Done!");
}
#pragma config(Sensor, port3, leftLineFollower, sensorVexIQ_ColorGrayscale)
#pragma config(Sensor, port4, rightLineFollower, sensorVexIQ_ColorGrayscale)
#pragma config(Sensor, port8, touchLED, sensorVexIQ_LED)
#pragma config(Sensor, port11, crossingLine, sensorVexIQ_ColorGrayscale)
#pragma config(Motor, motor1, leftBucket, tmotorVexIQ, PIDControl)
#pragma config(Motor, motor2, leftScoop, tmotorVexIQ, PIDControl)
#pragma config(Motor, motor5, rightScoop, tmotorVexIQ, PIDControl)
#pragma config(Motor, motor6, rightBucket, tmotorVexIQ, PIDControl)
#pragma config(Motor, motor7, leftWheel, tmotorVexIQ, PIDControl)
#pragma config(Motor, motor12, rightWheel, tmotorVexIQ, PIDControl)

// Include the common code used in both autonomous and team skill
#include "CommonCode.c"
#include "CommonDriver.c"

// Initialize the different parts of the robot
void InitializeTeamControl()
{
    // Line followers
    leftLineFollowerBlack = 0;
    leftLineFollowerWhite = 0;
    rightLineFollowerBlack = 0;
    rightLineFollowerWhite = 0;

    // Crossing line
    crossingLineBlack = 0;
    crossingLineWhite = 0;
    degreesPerMillimeter = 1.81;

    // Is the battery okay?
    if(IsBatteryOK() == false)
    {
        // No. Wait to hear the sound
        wait1Msec(3000);
    }

    // Initialize the scoop
    InitializeScoop(55);

    // Calibrate light sensors
    CalibrateLightSensors();

    // Initialize the bucket
    InitializeBucket(75);

    // Back up until sense the line
    MoveUntilLine(false, 30, crossingLine);

    // Move off the line so the light sensors will be off the line
    Move(false, 150, 30);

    // Move the robot over so we line up with the edge of the line
    TurnOneWheel(false, 30, 30, true);
    TurnOneWheel(true, 30, 30, true);

    // Backup so the line followers get most of the line
    Move(false, 200, 30);
FollowLineUntilCrossingLine(10, leftLineFollower, true);

TurnOneWheel(true, 13, 10, false);
TurnOneWheel(false, 13, 10, false);

Move(true, 45, 10);

SetScoopPosition(22, 30, false);

SetBucketPosition(0, 30, true);

// This is the main task that runs first

task main()
{
    // Initialize the main
    InitializeMain();

    // Clear the debug stream and write a message
    clearDebugStream();
    Write("Program Starting");

    // Make the LED display Red
    // This means we are waiting for the LED to be touched
    setTouchLEDRGB(touchLED, 255, 0, 0);
    setTouchLED BlinkTime(touchLED, 0, 0);

    // Wait until user touches the LED touch sensor
    Write(""");
    Write("Pre-initialize mode");
    Write("Waiting for touch");
    WaitUntilTouchAndRelease();
    Write("Touched");

    // Make the LED display flashing Yellow
    // This means the robot is initializing
    setTouchLEDRGB(touchLED, 255, 255, 0);
    setTouchLEDBlinkTime(touchLED, 10, 10);

    // Initialize the robot
    InitializeTeamControl();

    // Make the LED display flashing Blue
    // This means the robot is running
    setTouchLEDRGB(touchLED, 0, 0, 255);
    setTouchLEDBlinkTime(touchLED, 10, 10);

    // Run the program
    RunDriverControl();

    // Turn the light off
    setTouchLEDRGB(touchLED, 0, 0, 0);
File: C:\Data\RobotC Programs\Knights-Team.c

Write("Program Done!");
}
```c
#include "CommonCode.c"
#include "CommonDriver.c"

void InitializeRestart()
{
    // Line followers
    leftLineFollowerBlack = 0;
    leftLineFollowerWhite = 0;
    rightLineFollowerBlack = 0;
    rightLineFollowerWhite = 0;

    // Crossing line
    crossingLineBlack = 0;
    crossingLineWhite = 0;
    degreesPerMillimeter = 1.81;

    // Initialize the scoop
    InitializeScoop(0);

    // Initialize the bucket
    InitializeBucket(0);
}

// This is the main task that runs first.
__attribute__((interrupt)) void main()
{
    // Initialize the main
    InitializeMain();

    // Clear the debug stream and write a message
    clearDebugStream();
    Write("Program Starting");

    // Make the LED display Red
    // This means we are waiting for the LED to be touched
    setTouchLEDRgb(touchLED, 255, 0, 0);
    setTouchLEDLinkTime(touchLED, 0, 0);

    // Wait until user touches the LED touch sensor
    Write("*");
    Write("Pre-initialize mode");
    Write("Waiting for touch");
    WaitForTouchAndRelease();
    Write("Touched");
}```
// Make the LED display flashing Yellow
// This means the robot is initializing
setTouchLEDGRGB(touchLED, 255, 255, 0);
setTouchLEDBlinkTime(touchLED, 10, 10);

// Initialize the robot
InitializeRestart();

// Make the LED display flashing Blue
// This means the robot is running
setTouchLEDGRGB(touchLED, 0, 0, 255);
setTouchLEDBlinkTime(touchLED, 10, 10);

// Run the program
RunDriverControl();

// Turn the light off
setTouchLEDGRGB(touchLED, 0, 0, 0);
Write("Program Done!");
// Remove small joy stick values
int Trim(int joystick)
{
    // Is joy stick less then ten?
    if(abs(joystick) < 10)
    {
        // Yes. Return zero so the robot won't move
        return 0;
    } else
    {
        // No. Just return the value
        return joystick;
    }
}

// Run driver control code
void RunDriverControl()
{
    // Define variables
    int straightSpeed;
    int turnSpeed;
    int rightSpeed;
    int leftSpeed;
    int bucketSpeed;
    int scoopSpeed;
    int bucketMinDegrees;
    int bucketMaxDegrees;
    int scoopMinDegrees;
    int scoopMaxDegrees;
    int bucketDegrees;
    int scoopDegrees;
    int bucketDumpDegrees;
    bool gotBalls;
    bool dumpAndHang;

    // Initialize variables
    bucketMinDegrees = 0;
    bucketMaxDegrees = 1425;
    bucketDumpDegrees = 1100;
    scoopMinDegrees = 0;
    scoopMaxDegrees = 2000;
    gotBalls = false;
    dumpAndHang = false;

    // Set the motor strength to the maximum
    setMotorCurrentLimit(leftWheel,512);
    setMotorCurrentLimit(rightWheel,512);
    setMotorCurrentLimit(leftBucket,512);
    setMotorCurrentLimit(rightBucket,512);
    setMotorCurrentLimit(leftScoop,512);
    setMotorCurrentLimit(rightScoop,512);

    // Loop forever
    while(true)
    {
        // Are we getting the goal balls?
        if(gotBalls == false && getJoystickValue(BtnEUp) == 1 && getJoystickValue(}
File: C:\Data\RobotC Programs\CommonDriver.c

{
    // Yes. Back up so we can get the goal balls
    Move(false, 50, 100);

    // Set the scoop down so we can get the goal balls
    SetScoopPosition(4, 90, false);

    // Position the robot to get the first ball
    TurnOneWheel(false, 13, 100, false);

    // Move forward to get the goal balls
    Move(true, 200, 100);

    // Make sure we don't execute this again
    gotBalls = true;
}

// Are we dumping and hanging?
else if(dumpAndHang == false && getJoystickValue(BtnEDown) == 1 && getJoys
{
    // Turn cancel mode on
    SetCancelMode(true);

    // Start lifting the bucket
    SetBucketPosition(73, 100, true);

    // Yes. Back up to the goal until we hit the goal
    Move2(false, 500, 90, true);

    // Lift the bucket so the balls can dump out
    SetBucketPosition(100, 100, true);

    // Wait so the balls can exit the bucket
    WaitWithCancel(500);

    // Start the bucket going down so we can get the balls from the scoop
    SetBucketPosition(5, 100, false);

    // Move forward so the bucket doesn't hit the goal
    Move(true, 250, 90);

    // Wait until the bucket gets most of the way down
    WaitUntilBucketPosition(35, false);

    // Start lifting the scoop
    SetScoopPosition(140, 100, true);

    // Start the scoop coming down
    SetScoopPosition(30, 100, false);

    // Start lifting the bucket
    SetBucketPosition(73, 100, false);

    // Wait until the bucket can clear the goal
    WaitUntilBucketPosition(65, true);

    // Back up to the goal until we hit the goal
    Move2(false, 275, 90, true);
// Lift the bucket until the balls dump out
SetBucketPosition(100, 100, true);

// Wait so the balls can exit the scoop
WaitWithCancel(500);

// Move forward to the turning position
Move(true, 50, 100);

// Turn towards the hanging bar
TurnOneWheel(false, 55, 100, true);
Move(false, 100, 100);
TurnOneWheel(false, 65, 100, true);

// Start the scoop coming down
SetScoopPosition(5, 100, false);

// Move forward to the hanging bar until we hit the hanging bar
Move2(true, 600, 90, true);

// Move the bucket down so we can hang.
SetBucketPosition(0, 100, true);

// Turn cancel mode off
SetCancelMode(false);

// Get the straight value from the joystick
straightSpeed = Trim(getJoystickValue(ChA));

// Is the slow straight button pushed
if (getJoystickValue(BtnFDown) == 1) {
    // Yes. Reduce the straight speed
    straightSpeed = straightSpeed / 3;
}

// Get the turn value from the joystick
turnSpeed = Trim(getJoystickValue(ChC));

// Is the slow turn button pushed
if (getJoystickValue(BtnEDown) == 1) {
    // Yes. Reduce the turn speed
    turnSpeed = turnSpeed / 4;
}

// Calculate the speed for each of the wheels
rightSpeed = straightSpeed - (turnSpeed / 2);
leftSpeed = straightSpeed + (turnSpeed / 2);

// Set the speed on the motor
setMotorSpeed(rightWheel, rightSpeed);
setMotorSpeed(leftWheel, leftSpeed);

// Get the bucket degrees
bucketDegrees = getMotorEncoder(leftBucket);
// Is the bucket too high?
if (bucketDegrees >= bucketMaxDegrees)
{
    // Yes. Only enable the down button
    bucketSpeed = (0 - getJoystickValue(BtnLDown)) * 100;
}

// Is the dump button pushed and is the bucket less than our dump position
else if (getJoystickValue(BtnFUp) == 1 && bucketDegrees < bucketDumpDegrees)
{
    // Yes. Make the bucket go up
    bucketSpeed = 100;
}

// Is the bucket too low?
else if (bucketDegrees <= bucketMinDegrees)
{
    // Yes. Only enable the up button
    bucketSpeed = (getJoystickValue(BtnLUp) - 0) * 100;
}

// The bucket in range
else
{
    // No. Get bucket speed from joystick for both up and down buttons
    bucketSpeed = (getJoystickValue(BtnLUp) - getJoystickValue(BtnLDown)) * 10;
}

// Tell the bucket what speed to go
setMotorSpeed(leftBucket, bucketSpeed);
setMotorSpeed(rightBucket, bucketSpeed);

// Get the scoop degrees
scoopDegrees = getMotorEncoder(leftScoop);

// Is the scoop too high?
if (scoopDegrees >= scoopMaxDegrees)
{
    // Yes. Only enable the down button
    scoopSpeed = (0 - getJoystickValue(BtnRDown)) * 100;
}

// Is the scoop too low?
else if (scoopDegrees <= scoopMinDegrees)
{
    // Yes. Only enable the up button
    scoopSpeed = (getJoystickValue(BtnRUp) - 0) * 100;
}

// The scoop in range
else
{
    // Get scoop speed from joystick for both up and down buttons
    scoopSpeed = (getJoystickValue(BtnRUp) - getJoystickValue(BtnRDown)) * 10;
}

// Tell the scoop what speed to go
setMotorSpeed(leftScoop, scoopSpeed);
ssetMotorSpeed(rightScoop, scoopSpeed);

// Wait for a short period of time
wait1Msec(1);
// Define variables
float startTime;
int crossingLineBlack;
int crossingLineWhite;
float degreesPerMillimeter;
int leftLineFollowerBlack;
int leftLineFollowerWhite;
int rightLineFollowerBlack;
int rightLineFollowerWhite;
bool canCancel;
bool cancelCommand;

// Initialize the main
void InitializeMain()
{
  // Initialize the variables
  canCancel = false;
  cancelCommand = false;
}

// Turns the cancel mode on or off
void SetCancelMode(bool cancelMode)
{
  // Save the cancel value
  canCancel = cancelMode;
  cancelCommand = false;
}

// Cancel all remaining commands
void CancelCommands()
{
  // Cancel commands
  cancelCommand = true;
}

// Check to see if the command is cancelled
bool IsCommand Cancelled()
{
  bool cancelled;
  cancelled = false;

  // Have we cancelled before?
  if(cancelCommand == false)
  {
    // No.
    // Can we cancel and did we juse push our joystick button?
    if(canCancel == true && getJoystickValue(BtnUp) == 1)
    {
      // Yes. Start cancel command
      cancelCommand = true;
      cancelled = true;
    }
  }
  else
  {
    // Yes. Return true
    cancelled = true;
  }
}
File: C:\Data\RobotC Programs\CommonCode.c

} // Return the test results
return cancelled;

} // Wait for a period of time and allow it to be cancelled
void WaitWithCancel(int milliseconds)
{
    // Declare variables
    bool done;
    long startTime;
    long millisecondsWaited;

    // Get the current time
    startTime = nPgmTime;

    // Loop until done
    done = false;
    while (done == false)
    {
        // Should we cancel the wait?
        if (IsCommandCancelled() == true)
        {
            // Yes. Get out of the loop
            done = true;
            continue;
        }

        // Calculate how many milliseconds have we waited
        millisecondsWaited = nPgmTime - startTime;

        // Are we done waiting?
        if (millisecondsWaited >= milliseconds)
        {
            // Yes. Get out of the loop
            done = true;
        }
        else
        {
            // No. Wait
            wait1Msec(1);
        }
    }
}

// Write a message to the debug stream
void Write(string *text)
{
    float seconds = nPgmTime / 1000.0;
    writeDebugStreamLine("%f: %s", seconds, text);
}

// Wait until a motor begins moving
void WaitUntilMotorMoving(tMotor motor)
{
    // Is the motor moving?
    }
while (!IsCommandCancelled() == false && getMotorMoving(motor) == 0)
{
    // No, wait
    wait1Msec(1);
}

// Wait until a motor stops moving
void WaitUntilMotorStopsMoving(tMotor motor)
{
    // Is the motor moving?
    while (IsCommandCancelled() == false && getMotorMoving(motor) == 1)
    {
        // Yes, wait
        wait1Msec(1);
    }
}

bool IsBatteryOK()
{
    // Define Variables
    float minimumBatteryVoltage = 7.75; // minimum millivolts

    // Get the voltage of the battery
    float batteryVoltage = nImmediateBatteryLevel / 1000.0; // current battery

    // Is Battery voltage lower than minimum?
    if (batteryVoltage < minimumBatteryVoltage)
    {
        // Yes. Send Warning to screen & beep alarm.
        bPlaySounds = true; // Turn on sound.
        nVolume = 10;
        displayTextLine(1, "*** WARNING *** ");
        displayTextLine(2, "Battery LOW (%2.1f)", batteryVoltage);
        playRepetitiveSound(soundSiren2, 300);
        return false;
    }
    else
    {
        // No. Battery is OK.
        displayTextLine(1, "Voltage = %2.1f OK", batteryVoltage);
        return true;
    }
}

void Move2(bool moveForward, float millimeters, int speed, bool waitUntilDone)
{
    // Set the encoders to zero
    resetMotorEncoder(leftWheel);
    resetMotorEncoder(rightWheel);

    // Convert millimeters to degrees
    float degrees = millimeters * degreesPerMillimeter;

    // Are we moving backwards?
    if (moveForward == false)
File: C:\Data\RobotC Programs\CommonCode.c

```c
{
    // Yes
    // To move the robot backwards, the degrees and speed need to be negative
    degrees = degrees * -1;
    speed = speed * -1;
}
writeDebugStreamLine("Degrees %d", degrees);

// Tell the robot to move
setMotorTarget(leftWheel, degrees, speed);
setMotorTarget(rightWheel, degrees, speed);

// Should we wait?
Write("should we wait?");
if(waitUntilDone == true)
{
    Write("Yes");
    Write("Waiting until moving");
    // Yes, Wait until wheel starts moving
    WaitUntilMotorMoving(leftWheel);
    WaitUntilMotorMoving(rightWheel);
    Write("Waiting until done moving");
    // Wait until motors are done moving
    // waitUntilMotorStop(leftWheel);
    // waitUntilMotorStop(rightWheel);
    WaitUntilMotorStopsMoving(leftWheel);
    WaitUntilMotorStopsMoving(rightWheel);
    Write("Done moving");
}
else
{
    Write("No");
}

// Reset time and display time work together
void ResetTimer()
{
    // Save the current time
    startTime = nPgmTime;
}

void DisplayTimer()
{
    float seconds = (nPgmTime - startTime) / 1000.0;
    displayTextLine(4,"Time = %2.1f seconds", seconds);
}

// Check if robot is moving
bool IsRobotMoving()
{
    // Check to see if any wheel motor is moving
    return getMotorMoving(leftWheel) == 1 ||
           getMotorMoving(rightWheel) == 1;
}
// Set the robot speed
void SetRobotSpeed(int speed)
{
  // Set the speed for each motor
  setMotorSpeed(leftWheel, speed);
  setMotorSpeed(rightWheel, speed);
}

// Wait until the crossing line is crossed
void WaitUntilLineCrossed(tSensors lightSensor)
{
  int lightSensorValue;
  bool done;
  int lineEdgeLightValue;

  // Calculate the light value of the edge of a line
  // Use the crossing line light sensor?
  if(lightSensor == crossingLine)
  {
    // Yes.
    // Using a crossing line light sensor
    lineEdgeLightValue = (crossingLineBlack + crossingLineWhite) / 2;
  }

  // Use the left line follower?
  else if(lightSensor == leftLineFollower)
  {
    // Yes.
    // Using a left line follower
    lineEdgeLightValue = (leftLineFollowerBlack + leftLineFollowerWhite) / 2;
  }

  // Use the right line follower?
  else if(lightSensor == rightLineFollower)
  {
    // Yes.
    // Using a right line follower
    lineEdgeLightValue = (rightLineFollowerBlack + rightLineFollowerWhite) / 2;
  }

  // Loop until done
  done = false;
  while(done == false)
  {
    // Get the value of the light sensor
    lightSensorValue = getColorGrayscale(lightSensor);

    writeDebugStreamLine("%d,%d", lightSensorValue, lineEdgeLightValue);

    // Did the light sensor detect the line?
    if(lightSensorValue < lineEdgeLightValue)
    {
      // Yes. Get out of the loop
      done = true;
    }
  }
else
{
    // No it did not detect the line. wait a bit
    wait1Msec(1);
}

// Move until we cross a line
void MoveUntilLine(bool moveForward, int speed, tSensors lightSensor)
{
    // Are we moving backwards?
    if(moveForward == false)
    {
        // Yes. Make the speed negative so the robot will go backwards
        speed = speed * -1;
    }

    // Set the robot speed
    SetRobotSpeed(speed);

    // Wait until we cross the line
    WaitUntilLineCrossed(lightSensor);

    // Set the robot speed
    SetRobotSpeed(0);
}

// Calculate the streaming average of a number
float CalculateAverage(float currentAverage, float number, float count)
{
    // Define variables
    float average;
    float total;

    // Compute average
    total = (currentAverage * (count - 1)) + number;
    average = total / count;

    // Return the average
    return average;
}

// Follow the line until the crossing line sensor senses a crossing line
void FollowLineUntilCrossingLine(int speed, tSensors lightSensor, bool rightSensor)
{
    bool done;
    float Kp;
    float lineFollowerSensor;
    float error;
    float turn;
    float rightSpeed;
    float leftSpeed;
    float lineFollowerTarget;
    float crossingLineTarget;
    float crossingLineColorSensor;
    float lineFollowerRange;
    float lineFollowerTargetPercent;
float crossingLineTargetPercent;

// Initialize variables
lineFollowerTargetPercent = 0.7;
Kp = 0.07;
crossingLineTargetPercent = 0.25;

// Calculate the light value of the target of the line for the line follower

// Use the left line follower?
if(lightSensor == leftLineFollower)
{
    // Yes.
    // Using a left line follower
    lineFollowerRange = leftLineFollowerWhite - leftLineFollowerBlack;
    lineFollowerTarget = leftLineFollowerBlack + (lineFollowerRange * lineFollowerSensor);
}

// Use the right line follower?
else if(lightSensor == rightLineFollower)
{
    // Yes.
    // Using a right line follower
    lineFollowerRange = rightLineFollowerWhite - rightLineFollowerBlack;
    lineFollowerTarget = rightLineFollowerBlack + (lineFollowerRange * lineFollowerSensor);
}

crossingLineTarget = ((crossingLineWhite - crossingLineBlack) * crossingLine); done = false;
while(!done)
{
    // Get the value of the light sensor
    lineFollowerSensor = getColorGrayscale(lightSensor);

    // Calculate the error
    error = lineFollowerTarget - lineFollowerSensor;

    // Calculate our turn
    turn = error * Kp;

    // Calculate the drive speeds
    // Follow the right side of the line?
    if(rightSideOfLine == true)
    {
        // Yes.
        leftSpeed = speed + turn;
        rightSpeed = speed - turn;
    }
    else
    {
        // No.
        leftSpeed = speed - turn;
        rightSpeed = speed + turn;
    }

    // Tell the motors their speed
    setMotorSpeed(leftWheel, leftSpeed);
setMotorSpeed(rightWheel, rightSpeed);

    // Get the value of the crossing line sensor and its average
    crossingLineLightSensor = getColorGrayscale(crossingLine);

    // Did the light sensor detect the line?
    if(crossingLineLightSensor < crossingLineTarget)
    {
        // Yes. Stop the motors
        SetRobotSpeed(0);

        // Get out of the loop
        done = true;
    }
    else
    {
        // No it did not detect the line. Wait a bit
        wait1Msec(1);
    }

}  // End loop

// Turn the robot using one wheel
void TurnOneWheel(bool turnLeft, int robotDegrees, int speed, bool forward)
{
    // Define Variables
    float wheelDegrees;
    bool done;
    tMotor movingWheel;
    tMotor stoppedWheel;
    float currentRobotDegrees;
    float robotDegreesPerWheelDegree;

    // Initialize variables
    robotDegreesPerWheelDegree = 0.13;

    // Are we moving backwards?
    if(forward == false)
    {
        // Yes. Make the speed negative so the robot will turn backwards
        speed = speed * -1;
    }

    // Are we turning left?
    if(turnLeft == true)
    {
        // Yes. Set the right wheel as the moving wheel
        movingWheel = rightWheel;
        stoppedWheel = leftWheel;
    }
    else
    {
        // No. Set the left wheel as the moving wheel
        movingWheel = leftWheel;
        stoppedWheel = rightWheel;
    }

    // Reset the encoder to zero
resetMotorEncoder(movingWheel);

// Stop one wheel and turn the other wheel
setMotorSpeed(stoppedWheel, 0);
setMotorSpeed(movingWheel, speed);

// Loop until done
done = false;
while(IsCommandCancelled() == false && done == false)
{
    // Get our current wheel degrees
    wheelDegrees = getMotorEncoder(movingWheel);

    // Convert wheel degrees to robot degrees
    currentRobotDegrees = wheelDegrees * robotDegreesPerWheelDegree;

    writeDebugStreamLine("%.2lf, %.2lf", currentRobotDegrees, robotDegrees);

    // Have we turned enough?
    if(abs(currentRobotDegrees) > robotDegrees)
    {
        // Yes. Stop the motors
        SetRobotSpeed(0);

        // Get out of the loop
        done = true;
    }
    else
    {
        // No it did not turn enough. Wait a bit
        wait1Msec(1);
    }
}

// Turn the robot using two wheels
void TurnTwoWheels(bool turnLeft, int robotDegrees, int speed, bool forward)
{
    // Define Variables
    float wheelDegrees;
    bool done;
    float currentRobotDegrees;
    float robotDegreesPerWheelDegree;
    int rightSpeed;
    int leftSpeed;

    // Initialize variables
    robotDegreesPerWheelDegree = 0.13;

    // Are we moving backwards?
    if(forward == false)
    {
        // Yes. Make the speed negative so the robot will turn backwards
        speed = speed * -1;
    }

    // Are we turning left?
if (turnLeft == true) {
    // Yes. Turn the left wheel backwards and the left wheel forwards
    leftSpeed = speed * -1;
    rightSpeed = speed;
} else {
    // No, we are turning right. Turn the left wheel forwards and the right wheel backwards
    leftSpeed = speed;
    rightSpeed = speed * -1;
}

// Reset the encoders to zero
resetMotorEncoder(leftWheel);
resetMotorEncoder(rightWheel);

// Turn both motors on
setMotorSpeed(leftWheel, leftSpeed);
setMotorSpeed(rightWheel, rightSpeed);

// Loop until done
done = false;
while (done == false) {
    // Get our current wheel degrees which is the sum of the right and left wheel degrees
    wheelDegrees = abs(getMotorEncoder(rightWheel)) + abs(getMotorEncoder(leftWheel));
    // Convert wheel degrees to robot degrees
    currentRobotDegrees = wheelDegrees * robotDegreesPerWheelDegree;
    writeDebugStreamLine("%2.1f, %2.1f", currentRobotDegrees, robotDegrees);
    // Have we turned enough?
    if (currentRobotDegrees > robotDegrees) {
        // Yes. Stop the motors
        SetRobotSpeed(0);
        // Get out of the loop
        done = true;
    } else {
        // No it did not turn enough. Wait a bit
        wait1Msec(1);
    }
}

// Wait until the touch LED is touched and released
void WaitUntilTouchAndRelease() {
    // Sleep while the touch LED is not touched
    while (getTouchLEDValue(touchLED) == 0) {
        sleep(25);
    }
}
void SquareToLine(bool moveForward, int speed)
{
    // Define the variables
    float targetPercent;
    float targetRight;
    float targetLeft;
    bool rightDone;
    bool leftDone;
    float rightLightSensor;
    float leftLightSensor;

    // Initialize variables
    targetPercent = 0.25;

    // Are we moving backward?
    if (moveForward == false)
        // Yes. Make the speed negative so we can go backwards
        speed = speed * -1;

    // Start the wheels moving
    setMotorSpeed(leftWheel, speed);
    setMotorSpeed(rightWheel, speed);

    // Calculate targets
    targetRight = ((rightLineFollowerWhite - rightLineFollowerBlack) * targetPercent;
    targetLeft = ((leftLineFollowerWhite - leftLineFollowerBlack) * targetPercent;

    // Loop until square to line
    rightDone = false;
    leftDone = false;
    while(rightDone == false || leftDone == false)
    {
        // Is right sensor on the line?
        rightLightSensor = getColorGrayscale(rightLineFollower);
        if (rightLightSensor < targetRight)
        {
            // Yes. Stop the right wheel
            setMotorSpeed(rightWheel, 0);

            // Tell the loop the right wheel is done
            rightDone = true;
        }

        // Is left sensor on the line?
        leftLightSensor = getColorGrayscale(leftLineFollower);
if(leftLightSensor < targetLeft)
{
    // Yes. Stop the left wheel
    setMotorSpeed(leftWheel, 0);
    // Tell the loop the left wheel is done
    leftDone = true;
}
}

// Moving straight in millimeters
void Move(bool moveForward, float millimeters, int speed)
{
    // Define the variables
    float wheelDegrees;
    bool done;
    float currentMillimeters;
    float millimetersPerWheelDegree;

    // Initialize the variables
    millimetersPerWheelDegree = 0.56;

    // Set the encoders to zero
    resetMotorEncoder(leftWheel);
    resetMotorEncoder(rightWheel);

    // Are we moving backwards?
    if(moveForward == false)
    {
        // Yes
        // To move the robot backwards, the degrees and speed need to be negative
        speed = speed * -1;
    }

    // Tell the robot to move
    setMotorSpeed(leftWheel, speed);
    setMotorSpeed(rightWheel, speed);

    // Loop until done
    done = false;
    while(!IsCommandCancelled() == false && done == false)
    {
        // Get our current wheel degrees
        wheelDegrees = (getMotorEncoder(leftWheel) + getMotorEncoder(rightWheel))

        // Convert wheel degrees to robot degrees
        currentMillimeters = wheelDegrees * millimetersPerWheelDegree;

        writeDebugStreamLine("%.2f, %.2f", currentMillimeters, millimeters);

        // Have we moved enough?
        if(\abs{currentMillimeters} > millimeters)
        {
            // Yes. Stop the motors
            SetRobotSpeed(0);
        }
    }
}
// Get out of the loop
done = true;
}
else
{
    // No it did not turn enough. Wait a bit
    wait1Msec(1);
}
}

// Set the scoop to a certain position
void SetScoopPosition(float scoopDegrees, int speed, bool waitUntilDone)
{
    // Define variables
    float motorDegrees;
    float gearRatio;

    // Initialize variables
    gearRatio = 15.0;
    setMotorStrength(leftScoop, 512);
    setMotorStrength(rightScoop, 512);

    // Calculate our motor degrees
    motorDegrees = scoopDegrees * gearRatio;

    // Tell the scoop to go to its position
    setMotorTarget(leftScoop, motorDegrees, speed);
    setMotorTarget(rightScoop, motorDegrees, speed);

    // Wait until it gets to its position?
    if(IsCommandCancelled() == false && waitUntilDone == true)
    {
        // Yes. Wait until scoop starts
        WaitUntilMotorMoving(leftScoop);
        WaitUntilMotorMoving(rightScoop);

        // Wait until scoop stops
        WaitUntilMotorStop(leftScoop);
        WaitUntilMotorStop(rightScoop);
        WaitUntilMotorStopsMoving(leftScoop);
        WaitUntilMotorStopsMoving(rightScoop);
    }
}

// Set the bucket to a certain position
void SetBucketPosition(float bucketDegrees, int speed, bool waitUntilDone)
{
    // Define variables
    float motorDegrees;
    float gearRatio;

    // Initialize variables
    gearRatio = 15.0;
    setMotorStrength(leftBucket, 512);
    setMotorStrength(rightBucket, 512);
// Calculate our motor degrees
motorDegrees = bucketDegrees * gearRatio;

// Tell the bucket to go to its position
setMotorTarget(leftBucket, motorDegrees, speed);
setMotorTarget(rightBucket, motorDegrees, speed);

// Wait until it gets to its position
if(waitUntilDone == true)
{
    // Yes. Wait until bucket starts
    WaitUntilMotorMoving(leftBucket);
    WaitUntilMotorMoving(rightBucket);

    // Wait until bucket stops
    WaitUntilMotorStop(leftBucket);
    WaitUntilMotorStop(rightBucket);
    WaitUntilMotorStopsMoving(leftBucket);
    WaitUntilMotorStopsMoving(rightBucket);
}

// Wait until the scoop reaches a position
void WaitUntilScoopPosition(float scoopDegrees, bool movingUp)
{
    // Define variables
    float motorDegrees;
    float gearRatio;
    bool done;
    float currentMotorDegrees;

    // Initialize variables
    gearRatio = 15;

    // Calculate our motor degrees
    motorDegrees = scoopDegrees / gearRatio;

    // Loop until done
    done = false;
    while(done == false)
    {
        // Get how far the scoop motor has turned
        currentMotorDegrees = getMotorEncoder(leftScoop);

        writeDebugStreamLine("%d, %d", currentMotorDegrees, motorDegrees);

        // Have we moved enough?
        if((movingUp == true && currentMotorDegrees > motorDegrees) ||
           (movingUp == false && currentMotorDegrees < motorDegrees))
        {
            // Yes. Get out of the loop
            done = true;
        }
        else
        {
            // No it did not turn enough. Wait a bit
            wait1Msec(1);
        }
    }
}
// Wait until the bucket reaches a position
void WaitUntilBucketPosition(float bucketDegrees, bool movingUp)
{
    // Define variables
    float motorDegrees;
    float gearRatio;
    bool done;
    float currentMotorDegrees;

    // Initialize variables
    gearRatio = 15;

    // Calculate our motor degrees
    motorDegrees = bucketDegrees * gearRatio;

    // Loop until done
    done = false;
    while(!IsCommandCancelled() || done)
    {
        // Get how far the bucket motor has turned
        currentMotorDegrees = getMotorEncoder(leftBucket);

        writeDebugStreamLine("%d, %d", currentMotorDegrees, motorDegrees);

        // Have we moved enough?
        if((movingUp == true && currentMotorDegrees > motorDegrees) ||
           (movingUp == false && currentMotorDegrees < motorDegrees))
        {
            // Yes. Get out of the loop
            done = true;
        }
        else
        {
            // No it did not turn enough. Wait a bit
            waitMsec(1);
        }
    }
}

// Initialize the bucket
void InitializeBucket(int position)
{
    // The bucket must be initialized to zero in the down position.

    // Reduce the motor strength so it doesn't damage the bucket because
    // it is geared 25 to 12
    setMotorStrength(leftBucket, 100);
    setMotorStrength(rightBucket, 100);

    // Make the bucket go down towards the zero position
    setMotorSpeed(leftBucket, -40);
    setMotorSpeed(rightBucket, -40);
// wait until the motor starts moving
WaitUntilMotorMoving(leftBucket);
WaitUntilMotorMoving(rightBucket);

// Stop the motor when bucket hits the ground
waitUntilMotorStop(leftBucket);
waitUntilMotorStop(rightBucket);

// Resets the motor position to zero
resetMotorEncoder(leftBucket);
resetMotorEncoder(rightBucket);

// Turn motors power to maximum level
setMotorStrength(leftBucket, 512);
setMotorStrength(rightBucket, 512);

// Set the bucket to a starting position
SetBucketPosition(position, 30, true);

// Initialize the scoop
void InitializeScoop(int position)
{
    // The scoop must be initialized to zero in the down position.
    setMotorBrakeMode(leftScoop, motorCoast);
    setMotorBrakeMode(rightScoop, motorCoast);

    // Reduce the motor strength so it doesn't damage the scoop because
    // It is geared 15 to 1
    setMotorStrength(leftScoop, 512);
    setMotorStrength(rightScoop, 512);

    // Make the scoop go down towards the zero position
    setMotorSpeed(leftScoop, -40);
    setMotorSpeed(rightScoop, -40);

    // Wait until the motor begins moving
    WaitUntilMotorMoving(leftScoop);
    WaitUntilMotorMoving(rightScoop);

    // Stop the motor when scoop hits the ground
    waitUntilMotorStop(leftScoop);
    waitUntilMotorStop(rightScoop);

    // Resets the motor position to zero
    resetMotorEncoder(leftScoop);
    resetMotorEncoder(rightScoop);

    // Turn motors power to maximum level
    setMotorStrength(leftScoop, 512);
    setMotorStrength(rightScoop, 512);

    // Move the scoop up to legal position
    SetScoopPosition(position, 50, true);
}

// Calibrate light sensors
void CalibrateLightSensors()
{
    int lineFollowerValue = 0;
    int crossingLineValue = 0;

    // Initialize variables
    leftLineFollowerBlack = 256;
    leftLineFollowerWhite = -1;
    rightLineFollowerBlack = 256;
    rightLineFollowerWhite = -1;
    crossingLineBlack = 256;
    crossingLineWhite = -1;

    // Move forward slowly across the line
    Move2(true, 100, 10, false);

    // Wait until robot starts moving
    while(IsRobotMoving() == false)
    {
        // Do nothing
    }

    // Loop until done moving
    while(IsRobotMoving() == true)
    {
        // Get the value of the line follower light sensor
        lineFollowerValue = getGrayscaleValue(leftLineFollower);

        // Is it smaller than our black value?
        if(lineFollowerValue < leftLineFollowerBlack)
        {
            // Yes. Save the new black value
            leftLineFollowerBlack = lineFollowerValue;
        }

        // Is it larger than our white value?
        if(lineFollowerValue > leftLineFollowerWhite)
        {
            // Yes. Save the new white value
            leftLineFollowerWhite = lineFollowerValue;
        }

        // Get the value of the line follower light sensor
        lineFollowerValue = getGrayscaleValue(rightLineFollower);

        // Is it smaller than our black value?
        if(lineFollowerValue < rightLineFollowerBlack)
        {
            // Yes. Save the new black value
            rightLineFollowerBlack = lineFollowerValue;
        }

        // Is it larger than our white value?
        if(lineFollowerValue > rightLineFollowerWhite)
        {
            // Yes. Save the new white value
            rightLineFollowerWhite = lineFollowerValue;
        }
    }
}
void CalibrateRobotPosition()
{
    // Move the bucket up so we don't hit the big ball
    SetBucketPosition(30.0, 30, false);

    // Back up until sense the line
    MoveUntilLine(false, 30, crossingLine);

    // Move off the line so the light sensors will be off the line
    Move(false, 275, 30);

    // Move the robot over so we line up with the edge of the line
    TurnOneWheel(false, 30, 30, true);
    TurnOneWheel(true, 30, 30, true);

    // Backup so the line followers get most of the line
    Move(false, 100, 30);

    // Position the robot on the table by following the line until the crossing
    FollowLineUntilCrossingLine(10, leftLineFollower, true);

    // Move the robot over so we line up with the edge of the line
    TurnOneWheel(true, 13, 10, false);
    TurnOneWheel(false, 13, 10, false);

    // Move into a legal position
    Move(true, 45, 10);

    // Wait until the robot moves a distance
    void WaitUntilDistance(float millimeters)
    {
        bool done;
        float currentDegrees;
float currentMillimeters;

// Loop until done
done = false;
while (done == false)
{
    // Get how far the wheel has turned
    currentDegrees = getMotorEncoder(leftWheel);

    // Converting degrees to millimeters
    currentMillimeters = currentDegrees / degreesPerMillimeter;

    writeDebugStreamLine("%d, %d", currentMillimeters, millimeters);

    // Have we moved enough?
    if (abs(currentMillimeters) > millimeters)
    {
        // Yes. Get out of the loop
        done = true;
    }
    else
    {
        // No it did not turn enough. Wait a bit
        wait1Msec(1);
    }
}

// Initialize the different parts of the robot
void Initialize()
{
    // Line followers
    leftLineFollowerBlack = 0;
    leftLineFollowerWhite = 0;
    rightLineFollowerBlack = 0;
    rightLineFollowerWhite = 0;

    // Crossing line
    crossingLineBlack = 0;
    crossingLineWhite = 0;
    degreesPerMillimeter = 1.81;

    // Is the battery okay?
    if (IsBatteryOK() == false)
    {
        // No. Wait to hear the sound
        wait1Msec(1000);
    }

    // Initialize the bucket
    InitializeBucket(100);

    // Initialize the scoop
    InitializeScoop(30);

    // Calibrate light sensors
    CalibrateLightSensors();
File: C:\Data\RobotC Programs\CommonCode.c

    // Calibrate robot position
    CalibrateRobotPosition();

Project Design Notes

Rubric Summary

<table>
<thead>
<tr>
<th>Topic of Interest</th>
<th>Robotic navigation in Farming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sources</td>
<td>Farm Show Newspaper Internet</td>
</tr>
<tr>
<td>Expert</td>
<td>Mr. Terry Anderson – inventor of the “Driverless” tractor</td>
</tr>
<tr>
<td>Thesis Statement</td>
<td>Math is critical to safely operate the “Driverless” tractor by converting the sensor data to information that can drive decisions</td>
</tr>
<tr>
<td>Format</td>
<td>Interview by team</td>
</tr>
</tbody>
</table>

Presentation

Script

Characters:

1. Team Member 1 – Michael
2. Team Member 2 – Ben
3. Farmer – Maria
4. SPIRIT – Isaac

A-Ben: carry in Robot

A-Maria: carry in Ben’s boards

Michael: Hi! We are team 5509A, the VHS Knights. We discovered that robotics is revolutionizing the farming industry.

A-Maria: hold up “Farm Show” newspaper

Maria: I was just reading my “Farm Show” newspaper and found an article about the “Driverless” tractor invented by Mr. Terry Anderson. Our team called him, talked for over an hour, and learned so much!

Ben: WOW! That is amazing!!! Can you tell me more about it?

A-Isaac: hold up Board – SPIRIT & Expert

Isaac: I’m SPIRIT, one of the first “driverless” tractors. I have twin 202 horsepower engines, multiple computers and numerous sensors. So, Math is critical to convert the sensor data so I can safely operate in my farmer’s fields.

Michael: What sensors do you have and how is math used with them?

A-Maria: hold up Board – GPS/APS
Engineering Notebook
Project Design Notes

Maria: We use both Global and Area Positioning Systems to determine the exact position in the field. Using trilateration, this is accurate to inches.

Isaac: My GPS and APS tell me the time it takes for radio signals to travel from satellite or transponder to me. Then I multiply these values by the speed of light. This calculates my distance relative to the signal sources. Once I have at least 3 points of reference, I know exactly where I am.

Ben: Our robot doesn’t use trilateration. It uses 3 light sensors to follow and detect lines. This positions our robot within millimeters of accuracy.

Isaac: I use encoders in my motors and an accurate compass to help detect my speed and direction.

A-Michael: holds up Board – Encoder

Michael: Our robot also uses encoders in our motors. We input the distance we want our robot to move in millimeters. Our robot multiplies that number by encoder counts per millimeter. This calculates how many encoder counts to move.

A-Ben: hold up Board – Turn

Ben: We tried using a gyro sensor. But after several sensors failed, our team decided to remove them. We now accurately turn by braking one wheel and turning the other. We input the degrees we want to turn and our program calculates encoder degrees.

Maria: My SPIRIT uses Proportional, Integral, Derivative or PID control to accurately control its speed and positioning in the field.

Michael: We also programmed our robot to accurately follow a line using the Proportional control.

A-Ben: hold up Board – Gears

Isaac: I don’t need any gears because I have a simple electrical design.

Ben: We needed to use gears to help our motors lift our heavy bucket and scoop. We learned how to increase our torque by 25 times using a 25:1 gear ratio. But that was too slow!

Michael: To get more speed and still have enough torque, we added another motor to each lift, decreased to a 15:1 gear ratio, and added rubber bands to give extra torque. Now we can lift a full bucket or scoop 67% faster and also hang from the bar.

Maria: Well, there you have it! Sensors must be used to collect the data, but it is MATH that converts it to useful information.

A-ALL: hold up Boards – Sensors, +Math, =, Intelligence

ALL: Sensors + MATH = Intelligence!!!
Math is critical to convert sensor data so I can safely operate in my farmer's fields.

Our Expert
Mr. Terry Anderson
- Inventor of SPIRIT
- Owner of Automation Research Corporation

Global Positioning System (GPS) + Area Positioning System (APS) = Accurate to Inches

APS = 4 transponders in field and 2 on SPIRIT

Trilateration is determining a position by knowing the distance from at least 3 known points.
Encoders are in our SMART motors.

Distance in Millimeters
X Encoder Counts per Millimeter
= Encoder Counts

Turning

Brake

Turn

Turn Degrees
X Encoder Degrees per Turn Degrees
= Encoder Degrees
Engineering Notebook
Project Design Notes

Board 5 - Gearing

**Compound Gears**
Linked by Shared Axle
- 60 Tooth
- 36 Tooth
- 12 Tooth
- Rubber Bands

<table>
<thead>
<tr>
<th></th>
<th>First Design</th>
<th>Re-Design</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motors</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Gear Reduction</td>
<td>(60/12) X (60/12) = 25/1</td>
<td>(60/12) X (36/12) = 15/1</td>
<td></td>
</tr>
<tr>
<td>Speed</td>
<td>120 rpm/s / 25 = 4.8 rpm/s</td>
<td>120 rpm/s / 15 = 8.0 rpm/s</td>
<td>8.0/4.8 = 167%</td>
</tr>
<tr>
<td>Torque</td>
<td>0.414 N-m X 25 = 10.35 N-m X 1 motor = 10.35 N-m</td>
<td>0.414 N-m X 15 = 6.21 N-m X 2 motors = 12.42 N-m</td>
<td>12.42/10.35 = 120%</td>
</tr>
</tbody>
</table>

Board 6 - Sources of Information

**Sources of Information**

- "Farm Show" Newspaper And Book
- Numerous Websites
- Expert

Mr. Terry Anderson
- Inventor of SPIRIT
- Owner of Automation Research Corporation

Project - 5
Board 7 - Sensors

Sensors

Board 8 - +Math

+Math

Project - 6
Board 9 - Equals

Equals

Board 10 - Intelligence!

Intelligence!
Engineering Notebook
Project Design Notes

Sources of Information

Expert
Mr. Terry Anderson is the inventor of SPIRIT and owner of Automation Research Corporation. He spent the bulk of his career in Minnetonka, MN, where he founded the Automation Research Group and seven technology companies. We had a teleconference call with Mr. Anderson for over an hour learning about his design of SPIRIT and what he sees as advances in the Robotics industry.

Books and Newspapers
“Farm Show” Newspaper and Book

Websites
- How it works
- FarmShow.com
- AutonomousTractor.com
- WikiPedia.com
# VEX IQ ROBOT INSPECTION CHECKLIST

<table>
<thead>
<tr>
<th></th>
<th>INSPECTION ITEM</th>
<th>RULING #</th>
</tr>
</thead>
<tbody>
<tr>
<td>√</td>
<td>At the start of the match, the robot must only (1) touch the floor within the 11&quot; x 11&quot; square Starting Box (2) is no taller than 12&quot; and (3) no parts touch any other part of the field outside of the Starting Box.</td>
<td>&lt;R3&gt;</td>
</tr>
<tr>
<td></td>
<td>Robot is constructed ONLY from official components from the VEX IQ product line.</td>
<td>&lt;R5&gt;</td>
</tr>
<tr>
<td></td>
<td>The robot uses no more than 6 (SIX) VEX IQ motors.</td>
<td>&lt;R10&gt;</td>
</tr>
<tr>
<td></td>
<td>The robot uses no more than 1 (ONE) VEX IQ battery (7.2V).</td>
<td>&lt;R11&gt;</td>
</tr>
<tr>
<td></td>
<td>The robot uses no more than 1 (ONE) VEX IQ Robot Controller and 1 (ONE) VEX IQ Robot Brain.</td>
<td>&lt;R9&gt;</td>
</tr>
<tr>
<td></td>
<td>Team demonstrates that no VEX IQ robot parts have been modified.</td>
<td>&lt;R12&gt;</td>
</tr>
<tr>
<td></td>
<td>Robot does not have components that are intentionally detachable, pose a risk of entanglement, or pose a risk for field damage.</td>
<td>&lt;R13&gt;</td>
</tr>
<tr>
<td></td>
<td>Robot decorations are nonfunctional and do not affect performance.</td>
<td>&lt;R7&gt;</td>
</tr>
<tr>
<td></td>
<td>Robot displays the VEX IQ team identification number on at least two (2) opposing sides as 2&quot; x 3&quot; team number plates.</td>
<td>&lt;R2&gt;</td>
</tr>
</tbody>
</table>

Robot passes inspection when all check boxes are complete and this form includes inspector and team signatures.

Student team member accepts these inspection results and agrees that this robot was designed, built, and programmed by students, with little assistance from the adult mentor(s):

Team Student Signature: ___________________________ Date: ________________

Inspector Signature: ___________________________ Date: ________________