

**PHI OMEGA 4322**

**Engineering Notebook**

**PHI OMEGA**

**Engineering Notebook**

**Table of Contents**

**Team Bios:** page 3

* **Mentor Bios:** page 3

**Meetings:** page 4-20

**Hardware:** page 21-34

* **Cut Diagrams** page 22-23
* **Idea Sketches** page 24-27
* **Multiple Early Chassis** page 28-29
* **Dragster** page 30
* **Wiring Diagram** page 31
* **New hopper** page 32
* **AVHS FRC Pole Robot** page 33
* **New Arms** page 34
* **Software:** page 35-82
* **Programming Language Choosing** Page 35
* **Sensor Implementation** page 35
  + **pragmas** page 35
  + **Includes/Functions/Drivers** page 36
  + **initializeRobot** page 36
  + **Tele-Operation** page 36-43
  + **Tele-op Controls** Page 44
  + **Watch dog Failsafe** Page 44-45
  + **Autonomous** page 45-
    - **Functions** page 45-45
      * **Move** page 45-50
      * **Gyro turn** page 50-52
      * **Arc Turn** Page 52-55
      * **IR Align** Page 55-59
      * **Score** page 59-62
    - **auto programs** page#45-67
    - **DIFF B NP** Page 63-65
    - **DIFF B P** Page 66-68
    - **DIFF M P** Page 68-70
    - **DIFF RG M** Page 70-72
    - **DIFF RG SG** Page 72-74
    - **SAME B P** Page 74-77
    - **SAME C NP** Page 78-80
    - **SAME M D** Page 80-82

**Outreach:** page 82-89

* **Ocotillo Elementary School Education** page 82
* **Highland High school Career Day** page 82
* **NASA Education Open House/ Aero Institute Grand Reopening** page 82-84
* **2010 VEX Workshop** page 85
* **Lego Road Show** page 85
* **Super Science Saturday** page 85
* **High Desert FTC Scrimmage** page 86-87
* **Antelope Valley High School (FRC Team #2339)** page88
* **Las Vegas Elementary School Outreach** page 88
* **Fundraising**
* **NASA** page 89
* **Northrop Grumman** page 89
* **Sheriff Lee Baca** page 89
* **Jersey Mikes Sub’s** Page 89
* **Round Table Pizza** Page 89
* **California Soft Tissue Therapy** Page 89
* **Grace Chapel Ladies Te**a page 89
* **Garage Sale** page 89
* **Personal Contribution** page 89

**Team Members' Bios:**

Name: Wyatt Carr

Favorite job on team: Hardware, prototyping, Maintenance

Hobbies: Track, Cross Country, Tinkering

Interesting facts: I like to tinker and rebuild stuff. I once took apart my computer and put it back together without my parent’s permission and it worked.



Name: Grant Pickett

Favorite job on team: Prototyping

Hobbies: Writing, playing video games, playing guitar

Interesting facts: I am writing a series novels in my near nonexistent spare time.



Name: Aisha Rigert

Favorite job on team: Hardware, outreach, and organizing

Hobbies: music, Awana, working on my family’s music store web site

Interesting facts: I am a (deputy) black belt in Tae Kwon Do

Name: Dylan Stinson

Favorite job on team: programming, driving

Hobbies: play video games, watch anime, read fanfiction.

Interesting facts: I am Dyslexic and I’m deadly with a foil.

Name: Derek Strattan

Favorite job on team: Hardware, driving

Hobbies: air soft gunning, playing video games, collecting cars

Interesting facts: I love Ford cars but cannot stand Chevrolets

**Mentors' Bios**



Mr. Jon Stinson obtained his BSEE in 1988 and an MSTM in 2001. Mr. Stinson has worked in Aerospace since 1984. Mr. Stinson is currently an Associate Technical Fellow with Northrop Grumman. Mr. Stinson has been involved as a mentor with First Robotics for the last 8 years, 5 years of Lego League and 3 years of FTC.



Mr. Strattan obtained a BSME degree in 1985 and a BSEE degree in 1986 from the University of Tulsa. Mr. Strattan is currently employed by Northrop Grumman and has worked in the aerospace industry for 26 years. This is his first year as a mentor with First Robotics. Mr. Strattan’s senior design project team developed a six axis robotic arm driven by an Apple IIe computer in 1985.**Meetings:**

| Meeting #1 9/11/10 AR | |
| --- | --- |
| Goals  Learn about game  Start building field  Possibly brainstorm  Have fun | Accomplishments  Watched game videos and talked about game  Put together field and border  We will brainstorm on Tuesday  Have fun :) |

| Meeting #2 9/14/10 AR | |
| --- | --- |
| Goals  Discuss rules  Discuss game challenges and robot ideas  Start building field | Accomplishments  Discuss rules  Discuss game challenges and robot ideas   * Possibly corral rolling goals using robot body shape * Game challenges: getting to other side * Low center of gravity for better balance * Launch batons to other side of field for teammate to collect? * Getting into rolling goals: slide that rotates batons 90 degrees * Arm that comes down in front to herd 2nd rolling goal for end game * Little tube that batons go through, magnet sensor tests for magnet baton and turns on LED * Panels on servos, one flat, one slanted to guide batons   Start building rolling goals  Start laying out wood to cut  Cut all batons |

| Meeting #3 9/16/10 AR | |
| --- | --- |
| Goals  Build the field:   * Finish rolling goals * Work on cliffs * Work on bridges * Work on mountain * Start dispensers   Brainstorm   * Scoring * Defense   Set up and write in Engineering Notebook | Accomplishments  Field building:   * Finish rolling goals * Draw out wood to cut for dispensers * Cut and sand bridge support   Brainstorm autonomous strategy:  Top priority:  Doubler, magnetic baton, dump dispensers  Cross to the other side- handle either cliff or bridge- probably bridge  Should we pick up from ground or from dispensers? We think both  Not use sensors (except touch sensors) because they have problems?  Starting place affects strategy   * Get over to other side * Get all batons and doubler in low goal, then guard them * Get magnetic and doubler in autonomous * Magnetic: sensor that just runs down the middle of the dispenser * Store magnetic, toss others * Find dispenser(s) with magnetic batons in them * Drop something magnetic in middle of pile of batons and pick up magnet batons * Dispense batons and immediately throw or herd them into low goal * Use magnetic object right by chute that batons go through to take the magnet baton directly out of line * Get doubler in autonomous * Empty ALL dispensers * Balance on bridge * Search for magnet batons * Take rolling goal to other side and fill * Throw into low goal * Collect batons other alliance has dumped * Move batons across the center * Move our goals in front of their dispensers to block them * Move their rolling goals into their low goal – take into account what type of baton delivery system they have * Move our rolling goals into our low goal * Push batons into low goal via cliff * Put their rolling goals on our bridge just before end game   Robot Ideas  Baton collection/scoring   * Tray in front of robot * Robot that contains rolling goal and drops batons into itself into the goal * Pick up batons on the floor using conveyor * Rotate 5 batons at a time using servo, drop 1 down at a time * Moving pipe that rotates to drop into any goal location- goal is inside robot * Goal goes only ½-way into robot   Rotating arm to score batons   * Rotating arm- moves out using linear gear?- that puts batons into goal * Front of robot aligns rolling goal * Rotating arm is on a path that includes 2 close goals and the center one * Use linear gear for arm   Other   * Gyroscopic sensor to cut off autonomous when we're about to tip, or to back up   Questions for forum:  Is corralling the goal grasping, holding, or lifting?  Is touching a baton considered possessing it (driving over it etc.)  If you're holding/touching 1 baton that is touching 5 others, are you “possessing” 6 (to push a row of batons etc.)  Does balancing other team's goals on your bridge give you points?  Set up and write in Engineering Notebook |

|  |  |
| --- | --- |
| Meeting #4 9/20/10 DMS |  |
| Goals | Accomplishments |
| * Build field * brainstorm * prototype | * More than 90% done on Dispensers * Made some prototypes |

| Meeting #5 9/23/10 GP | |
| --- | --- |
| Goals  Finish field  Prototype a simple chassis to cross mountain  Brainstorm  Take apart last year's robot | Accomplishments  Field finished, finished baton dispensers  Wyatt and I built a bot that took the mountain  Brains effectively stormed Thumper Died |

| Meeting #6 9/28/10 GP | |
| --- | --- |
| Goals  Discuss sensor potential  Brain storming  Prototyping | Accomplishments  Discuss sensor potential  Ideasssss:  Baton Gathering :  Fan  Baton Channel  Batons out the top  Driving:  Omni-wheels  Gear shift  Baton Holding:  Focus on doubler and magnetic  Baton clip/cartridge  flag raiser  Scoring:  Pushing batons  Throwing  3 Joint arm  Dump Truck  Test Sensors  Gear shift prototype made  Transfer device for batons  Testing the sensors didn't go so well  omni-wheel chassis |

| Meeting #7 9/30/10 WWC | |
| --- | --- |
| Goals | Accomplishments |
| Prototype dispensing mechanisms | Wheel dispensing mechanism created  Spinning arm created |

| Meeting #8 10/5/10 TM | |
| --- | --- |
| Goals | Accomplishments |
| Brain storming / prototyping  Vaccum  Ziptie brush  & Claw | Build various prototypes  IMG_20101005_205114.jpg  Early meeting for game planning |

| Meeting #9 10/7/10 TM | |
| --- | --- |
| Goals | Accomplishments |
| Getting robot over the mountain  Dispensing batons | Got robot over the mountain (it flips over though)  Rebuilt the dispensing device  Got the servos to work properly |

| Meeting #10 10/12/10 DAS | |
| --- | --- |
| Goals | Accomplishments |
| Pick up batons off ground. Find a way to dispense batons. Find a way to score batons | Made a brush to pick up batons. Made a base to grab onto rolling goals. Made a way to dispense batons and drop them in the rolling goal |

| Meeting #11 10/14/10 AR | |
| --- | --- |
| Goals | Accomplishments |
| Build a working chassis: modify our old one or build a new one- maybe slanted? Needs lower center of gravity (old one can’t go over mountain) | Rebuilt old chassis so it worked, but it was too big  Built a new chassis that was the right size- about to test it |

| Meeting #12 10/21/10 GP | |
| --- | --- |
| Goals | Accomplishments |
| Build devices that can pick up batons, store them, then dispense them. | Brainstormed many ideas for pick up. Each seemed to have its own draw backs. We probably will develop something similar to a dust pan for pick up and fill the loader Built prototype hopper and dispenser.  In other news:  As a part of own tests, we seem to be having a problem dispensing and getting the batons. So far we’ve come up with several good ideas, we’ll just need to pick and test those prototypes. |

| Meeting #13 10/26/10 AR | |
| --- | --- |
| **Goals** | **Accomplishments** |
| Finalize or somewhat finalize our collecting design  Possibly, if this gets done, think more about strategy, especially end game strategy  Work on balancing on the bridge at end game  Driver practice with 2 bots on field  Dispensing wheel needs “fender” and skid plate | Discussed different ground collection/scoring ideas:  **Plain Dustpan**  Pros:   * Picks up quickly * Fast and easy to build   Cons:   * You always have to go against the wall * High likelihood of breaking hopper/getting too many batons * Doesn’t work on the side * Can’t sense magnet   **Waterwheel**  Pros:   * Can make it stop at 5 batons   Cons:   * Big device (including baton-holding areas) * Have to have a transfer device * Easily broken * Needs another DC motor (or continuous servo)   **Claw**  Pros:   * Precise (can also be a con) * Can’t fall out or break easily   Cons:   * Can’t build a very articulate one out of what we have * Hard to position to pick up * Can’t get batons continuously * Slow   **Pan and brush/encourager**  Pros:   * Works anywhere * Somewhat easy to make   Cons:   * Breaks easily if made of wrong material * Can’t sense magnet * Doesn’t work on the side   **More ideas:**  Pan with side-dispensing option  Pan with brush  Pan with back-dispensing option  Dustpan with hole in side, servo tips pan so they fall into goal  **5-baton clip is definite**  Slide to help direct batons to goal  Clip needs to be slightly wider than batons so they can self-align  Clip is wider at the front than at the back so they self-align  **Final idea:**  Pan made out of plastic  Rotates around 90 degrees so it’s gravity fed  Wheel pushes batons out the side into rolling goal |

| Meeting #14 10/28/10 DAS | |
| --- | --- |
| **Goals**  Make a hopper. Clean up bot. Practice driving. | **Accomplishments**  Made a new hopper. Cleaned up wires in bot. Had driver practice. |
|  |  |

| Meeting #15 11/2/10 AR | |
| --- | --- |
| **Goals**  Work on hopper, possibly driver practice | **Accomplishments**  Work on hopper and try to get it to work, including the wheel that pushes batons out of the hopper. |
|  |  |

| Meeting #16 11/4/10 AR | |
| --- | --- |
| **Goals**  Try to finish hopper and clean up. | **Accomplishments**  Work more on hopper. |
|  |  |

| Meeting #17 11/9/10 WWC | |
| --- | --- |
| **Goals**  Make robot fully functional  Autonomous programs  Hopper repair | **Accomplishments**  The robot is close to functionality  autonomous made only one though  hopper almost fully functional |
|  |  |

| Meeting #18 11/11/10 DMS | |
| --- | --- |
| **Goals**  Prototype encourager’s  Make functions | **Accomplishments**  Made two encouragers not working  Made functions move turn and arm |

| Meeting #19 11/16/10 GP | |
| --- | --- |
| **Goals**  Spend time planning strategies for the game  Incorporate continuous servos.  Add “encourager” to hopper. | **Accomplishments**  Remodeled robot to add structure and clean up wires.  Plan autonomous and tele-op game strategies.  Continuous servo broke. ☹ not our fault.  Added encourager not working yet.  Strategizing:  We need to plan out our time strategy for the 2:00 auto.  We need 2 strategies: 1 for if they dispense tons, then we should pick them up, another defense to block them dispensing, for if we dispense 1, if we dispense 2  What if another team can drag their rolling goals? Then we should pick up what they dispense.  This would also be good for if they just dispense a lot. Also if our alliance partner does this.  If our hopper doesn’t work very well, we could change it. But we are going to keep the dispensing arm. The hopper is too much precision, but it is way better than a claw. But putting an encourager would be better.  What if we were the only ones to dispense in autonomy, or at all?  We could possibly push their rolling goals into the corner, or into the low goal, or just rolling all over the place?  Score all in the low goal with doubler, then defend it |

| Meeting #20 11/18/10 DAS | |
| --- | --- |
| **Goals**  Driver practice  Work on engineering note book | **Accomplishments**  Had driver practice  Worked on hardware and software in engineering notebook |
|  |  |

| Meeting #21 11/23/10 GP | |
| --- | --- |
| **Goals**  Driver Practice  Fill out more of the notebook | **Accomplishments**  Driver Practice  Engineering notebook |
|  |  |

| Meeting #22 11/30/10 DMS | |
| --- | --- |
| **Goals**  Make the shielding | **Accomplishments**  Made the shielding |

| Meeting #23 12/2/10 WC | |
| --- | --- |
| **Goals**  Driver practice  Discuss scouting sheets  Discuss BOM  Set up sheet discuss | **Accomplishments**  Driver practice went very well we did contests to get our skill level up  Scouting sheets improved and revised  BOM is almost complete |
|  | Set up sheet added too |

| Meeting #24 12/7/10 AR | |
| --- | --- |
| **Goals**  Driver practice (everyone)  Engineering notebook- hardware, software, outreach, get up to date on meetings  Try to work on BOM, scout sheet, checklist | **Accomplishments**  Driver practice with and against 452  Massive hardware section update  Finish outreach section (almost- just 1 more picture to add)  Take home BOM, scout sheet, checklist with ideas to update |
|  |  |

| Meeting #25 12/9/10 AR | |
| --- | --- |
| **Goals**  Get ready for the LA tournament (this Saturday)  Driver practice | **Accomplishments**  Got ready for tournament  Driver practice |

| Meeting #26 12/14/10 GP | |
| --- | --- |
| **Goals**  Discuss LA competition  Build new hopper  Fix some issue in the autonomies | **Accomplishments**  Discussed  2 Cardboard Prototypes built  Progress made |
|  |  |

| Meeting #27 12/16/10 GP | |
| --- | --- |
| **Goals**  Create new hopper  Design more autonomies | **Accomplishments**  First prototype of new hopper in plastic made more features to be added  Concepts brainstormed, initial testing |

| Meeting #28 12/21/10 GP | |
| --- | --- |
| **Goals**  Further develop new hopper  Test autonomies  Driver practice | **Accomplishments**  Brush added and shield planned  Some testing  Some practice  (connection issues) |

| Meeting #29 12/28/10 DAS | |
| --- | --- |
| **Goals**  Driver practice  Work on autonomy | **Accomplishments**  Driver practice  Worked on autonomy |

| Meeting #30 1/3/10 DAS | |
| --- | --- |
| **Goals**  Driver practice  Work out strategy  Test autonomy | **Accomplishments**  Driver practice  Work out strategy  Test autonomy |

| Meeting #31 1/4/10 AR | |
| --- | --- |
| **Goals**  Test autonomies  Driver practice with 452 | **Accomplishments**  Tested autonomy  Found the rest of the field (so we can take it to San Diego) and adjusted it  Some driver practice with 452 |

| Meeting #32 1/6/10 AR | |
| --- | --- |
| **Goals**  Driver practice!!!  Practice picking up batons, scoring the doubler, and balancing in the end game as quickly as possible.  For driver practice, we invent contests that build our driving skill in a fun way, while concentrating on only one task. For example, we had a game where we had to start at a certain position and dispense as many batons as possible within one minute. This allows us to work on one skill (dispensing, collecting, or balancing) at a time. We find this to be a very fun and effective way to practice driving not only because we are working towards one goal at a time, but also because we are competing to get the best score! | **Accomplishments**  Driver practice:  1. We dumped out 15 batons, including the doubler, and we took turns driving for 2 minutes and trying to score and balance. This helped us get familiar with the robot’s collection mechanism and strategy for scoring.  2. We took turns trying to see if we could balance on the bridge in 10 seconds. The object was to try to pass the most levels, and each level was increasingly harder. The easy level started near the bridge, medium was farther, hard was even farther and had baton obstacles etc. This helped us get really fast at getting to and balancing on the bridge. One of our drivers actually started in the opposite low goal with 90 batons and 2 rolling goals in the way, and actually made it to the bridge (but didn’t balance.) Now *that’s* driver practice! |

|  |  |
| --- | --- |
| Meeting # 33 1/11/11 WWC | |
| Goals  Practice autonomies  Driver practice  Get most or all autonomies working at a high efficiency | Accomplishments  Most autonomies were used and are working well  All drivers practiced that were present most practice took place with 452  Most autonomies working at a high efficiency |

|  |  |
| --- | --- |
| Meeting # 34 1/13/11 WWC | |
| Goals  Practice autonomies  Driver practice | Accomplishments  Most autonomies were used and are working well  All drivers practiced that were present most practice took place with 452 |

|  |  |
| --- | --- |
| Meeting # 35 1/18/11 DMS | |
| Goals  Driver practice  Talk about san diego | Accomplishments  Drivers practiced with 452  Talk a lot of what we did right and wrong |

|  |  |
| --- | --- |
| Meeting # 36 1/20/11 DMS | |
| Goals  Driver practice  Fixed broken parts | Accomplishments  Drivers practiced played games to help  Bot fixed |

|  |  |
| --- | --- |
| Meeting # 37 1/25/11 GP | |
| Goals  Driver practice  prototype goal grabbing arm  test autonomy  Work on engineering notebook | Accomplishments  Drivers practiced  Finished a prototype goal grabbing arm  tested autonomy  Work on engineering notebook |

|  |  |
| --- | --- |
| Meeting # 38 1/27/11 DMS | |
| Goals  Practice diving  make new arm  finish notebook  pack for Vegas | Accomplishments  Driver practiced and drive team made  Made new are ad it works  Finished notebook  Packed for Vegas |

|  |  |
| --- | --- |
| Meeting # 39 2/1/11 AR | |
| Goals  Debrief after Las Vegas:  Talk about what worked and didn’t work  Discuss other teams’ strategies and how to combat them  Discuss what we need to do before St. Louis | Accomplishments  Discussed Las Vegas tournament  Made list of main goals/wish list for St. Louis  Discussed how to achieve these goals  Discussed EMP’s strategy of pushing rolling goals into the corner and guarding them. We want to move one of the rolling goals to the other side in autonomy so they can’t guard both goals at a time.  Wish list:  Grab rolling goal better in end game  Push rolling goal to other side in autonomy  Push rolling goal into low goal in autonomy  Score preloads more effectively  Sense magnet baton?  Dispense directly into hopper  Control arm motors from 1 motor port, replace tail motor with servos, and redo tail |

|  |  |
| --- | --- |
| Meeting # 40 2/3/11 DAS | |
| Goals  Mount two new servos for prototype on moving goal. | Accomplishments  Mounted two new servos. |

|  |  |
| --- | --- |
| Meeting # 41 2/8/11 DMS | |
| Goals  Brainstorm for moving rolling goals.  Tighten the robot | Accomplishments  Fix the robot  Found some good way to capture but have not decided on a final design |

|  |  |
| --- | --- |
| Meeting # 42 2/10/11 DAS | |
| Goals  Design a new prototype for moving rolling goals. | Accomplishments  Designed a new prototype and thought of another new one. |

|  |  |
| --- | --- |
| Meeting # 43 2/15/11 AR | |
| Goals  Re-mount the servos on the arms  Make new metal arms to go on the servos: we want L-shapes that, if used singly, can guide the rolling goal during teleop and that, if used together, can capture it during end game.  Driver practice | Accomplishments  We thought of ways we could open and shut our hopper (to replace the flap we already have.) We thought it would work better if the flap that keeps our hopper closed doesn’t swing down, because that is harder to line up.  The most feasible ideas they brainstormed included a flap that swivels to the side, a flap that swivels to the back, and two smaller flaps that open in opposite directions.  We re-mounted the small servo arms so that we can add something else in the middle of our robot. We are going to make L-shaped metal arms to go on the servos, but we have metal bars for right now. This works pretty well for guiding the goal. We can also push the goal over the mountain very easily without tipping it. |

|  |  |
| --- | --- |
| Meeting # 44 2/17/11 WWC | |
| Goals  Make grabber arms operational  Program grabber arm  Make armor | Accomplishments  The grabber arms were made operational  The grabber arms were programmed into tel.iop  Not accomplished |

|  |  |
| --- | --- |
| Meeting # 45 2/22/11 DAS | |
| Goals  Work on preload hopper.  Work on arms for rolling goals. | Accomplishments  Mounted preload arm  Made working arms for moving rolling goals. |

|  |  |
| --- | --- |
| Meeting # 46 2/24/11 DAS | |
| Goals  Remount NXT  Work on software programs | Accomplishments  Remounted the NXT will need to be moved again. Battery position is poor currently  Redid the preload scoring device  Added arm program |

|  |  |
| --- | --- |
| Meeting # 47 3/1/11 AR | |
| Goals  Rewire bot  Perfect preload scoring mechanism  Test watchdog program  Make shielding  Finalize brochure | Accomplishments  Finished preload scoring mechanism (a small cup on a servo arm which can hold 5 preloads and dump them accurately in autonomy)  Tested and finished the RobotC watchdog program for the robot; it stops the program running (teleop or autonomy) if the Samantha module disconnects. This will help us keep from burning out motors. Next: a Labview program that does the same thing.  Finished the brochure for St. Louis (except a current picture of the robot.) We are waiting to get a picture of the robot until the design is complete.  We also rewired the robot. |

|  |  |
| --- | --- |
| Meeting # 48 3/3/11 GP | |
| Goals  Mount front push plate  Driver practice | Accomplishments  Mounted front push plate  Practiced drivers |

|  |  |
| --- | --- |
| Meeting # 49 3/8/11 GP | |
| Goals  Fix side arms  Driver practice | Accomplishments  Side arms fixed  Practiced drivers |

|  |  |
| --- | --- |
| Meeting # 50 3/10/11 AR | |
| Goals  Redo the bot’s wiring  Driver practice  Make new shielding  Me (Aisha) and Derek: learn the basics of C so that next year we can do software  Redo front shield plate (it is not far forward enough, so when we push the rolling goals, they will hit the wheels instead of the shielding) | Accomplishments  Derek and I learned the basics of C from Dylan (our programmer). Next meeting, he is going to give us some programming tasks to see how much we remember.  We reorganized the bot’s wiring so the wires aren’t so messy. We have so many gears that it would be easy to shred our wires.  We started to design new shielding, but will have to finish next week (the programming lesson took most of our time.) We will also try to do the front shield plate next time. |

|  |  |
| --- | --- |
| Meeting # 51 3/15/11 WWC | |
| Goals  Driver Practice  Build shielding | Accomplishments  We practiced with our sister team  Design made for shielding |

|  |  |
| --- | --- |
| Meeting # 52 3/17/11 DMS | |
| Goals  Work on autonomys  Finish sheilding for bot  Driver practice | Accomplishments  Built sheilding for bot  Had driver practice  Fixed some of the problems with the autonomys |

|  |  |
| --- | --- |
| Meeting # 53 3/22/11 AR | |
| Goals  Driver practice (matches with 452 and games)  Fix rolling goal alignment piece on the front  Paint box that we transport the robot in  Work on autonomy | Accomplishments  Wrote one autonomy program (has bugs we need to work out)  Fixed the plastic piece on the front of the bot, which is used to align the rolling goals  Painted the robot’s box  Lots of driver practice! We ran matches with 452, and we also did games that build specific skills. Today, we practiced grabbing a rolling goal in 10 seconds. |

|  |  |
| --- | --- |
| Meeting # 54 3/24/11 DMS | |
| Goals  Driver practice  Fix anything wrong on bot  Tech team to install watchdog program | Accomplishments  Practiced driving  Replaced some motors  worked on telop  All teammates can install watchdog |

|  |  |
| --- | --- |
| Meeting # 55 3/29/11 DMS | |
| Goals  Driver practice (matches with 452 and games)  Talk adout what we are going to do at world | Accomplishments  Ran matches with 452  Start a checklist and scouting sheets  Started compass video |

|  |  |
| --- | --- |
| Meeting # 56 3/31/11 GP | |
| Goals  Driver practice | Accomplishments  Driver practice  Started to get pictures for compass |

|  |  |
| --- | --- |
| Meeting # 57 4/5/11 DAS | |
| Goals  Driver practice, stiffen arms that push down bridge. | Accomplishments  Had driver practice, stiffened the arms that push down the rolling goals. |

|  |  |
| --- | --- |
| Meeting # 58 4/7/11 WWC | |
| Goals  Driver practice | Accomplishments  Driver practice few hours  Recorded voiceover for compass video |

|  |  |
| --- | --- |
| Meeting # 58 4/12/11 AR | |
| Goals  Driver practice  Finish mini-batons to give out at Worlds | Accomplishments  Some driver practice  Finished about 280 (to be shared with PHI Alpha) mini-batons to give out at Worlds (instead of buttons)  Find music for compass video that we have the rights for |

|  |  |
| --- | --- |
| Meeting # 58 4/14/11 AR | |
| Goals  Driver practice  Finish video submission for Compass Award | Accomplishments  Mostly driver practice- we ran matches and tried to beat each other’s scores and score as much as possible  Finished the Compass Award video |

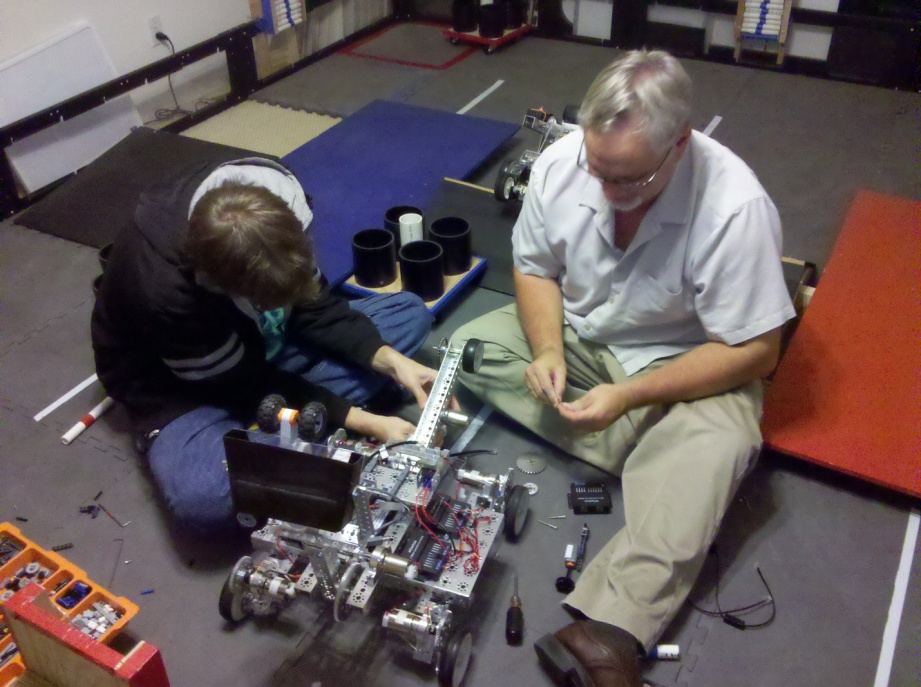
|  |  |
| --- | --- |
| Meeting # 59 4/19/11 DAS | |
| Goals  Finish mounting sheilding  Make a chain guard  Driver practice  Finish the bomb  Make new mount for NXT | Accomplishments  Finished mounting sheilding  Made a chain guard  Made new mount  Strengthen rolling goal arms  Drive practice sored a high score 147 with 452 |

|  |  |
| --- | --- |
| Meeting # 60 4/19/11 DAS | |
| Goals  Finish and proofread Engineering Notebook  Finish BOM and cut diagrams  Finish brochures  Pack for Worlds  Tighten and check over robot | Accomplishments  Finalized Notebook- added to hardware and software sections and checked over Notebook  Finished BOM and cut diagrams  Finished printing and folding brochures  Pack for Worlds  Checked over robot (including tightening) |

**Hardware:**

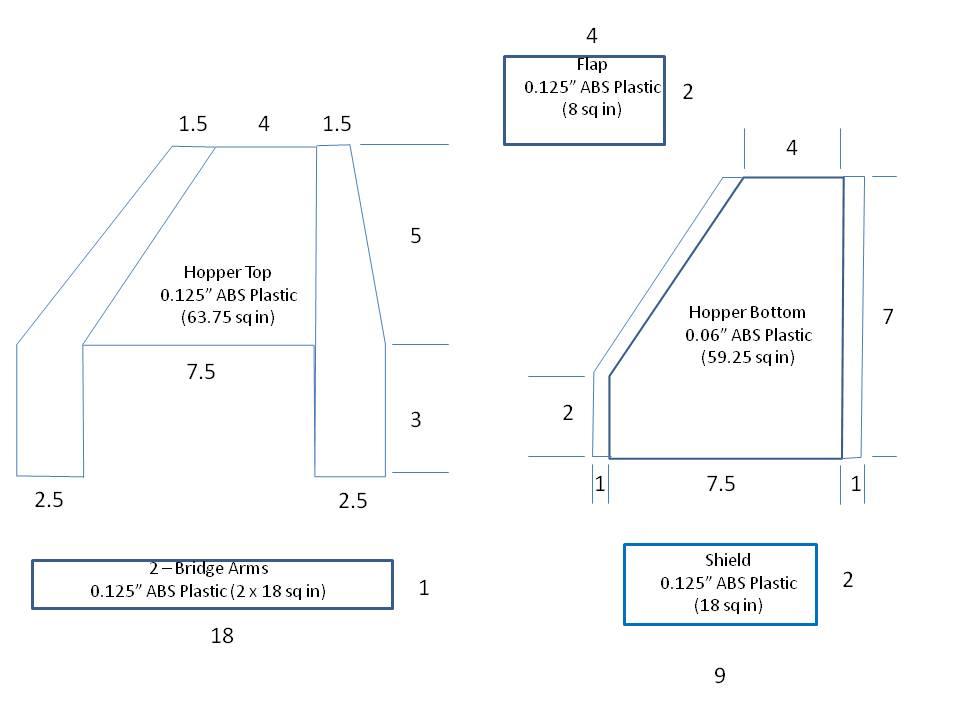


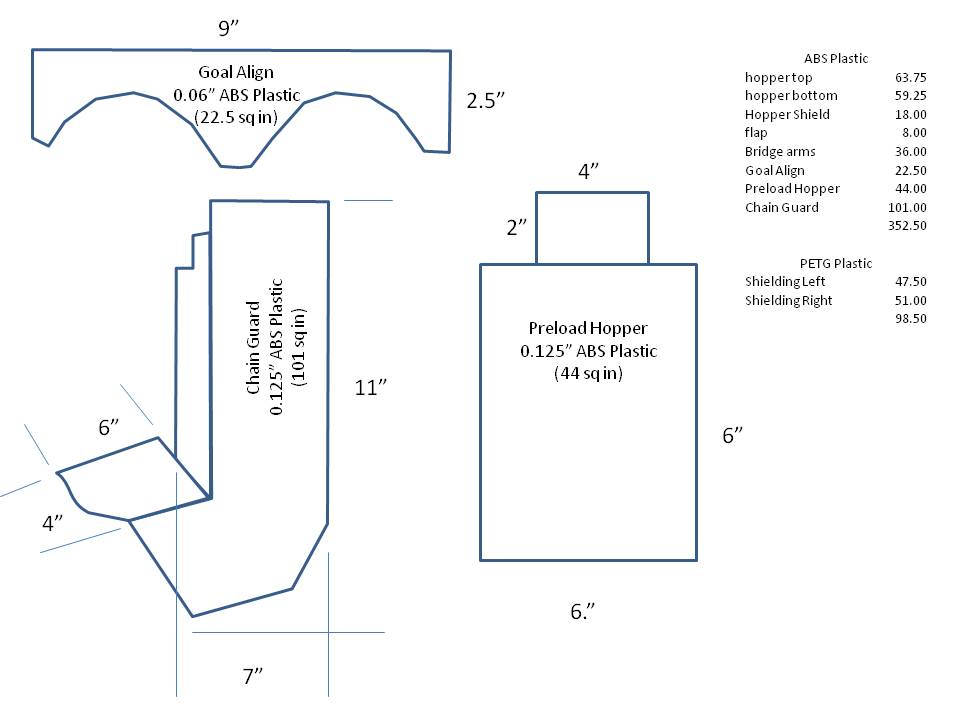
Old picture of the robot at Super Science Saturday at Joe Walker Middle School

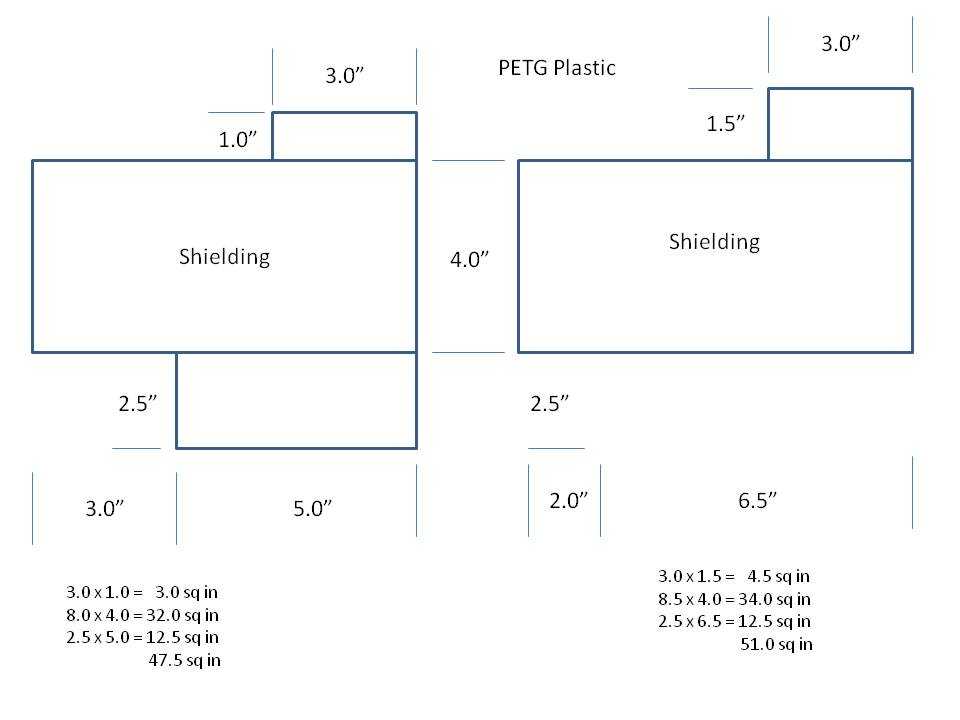


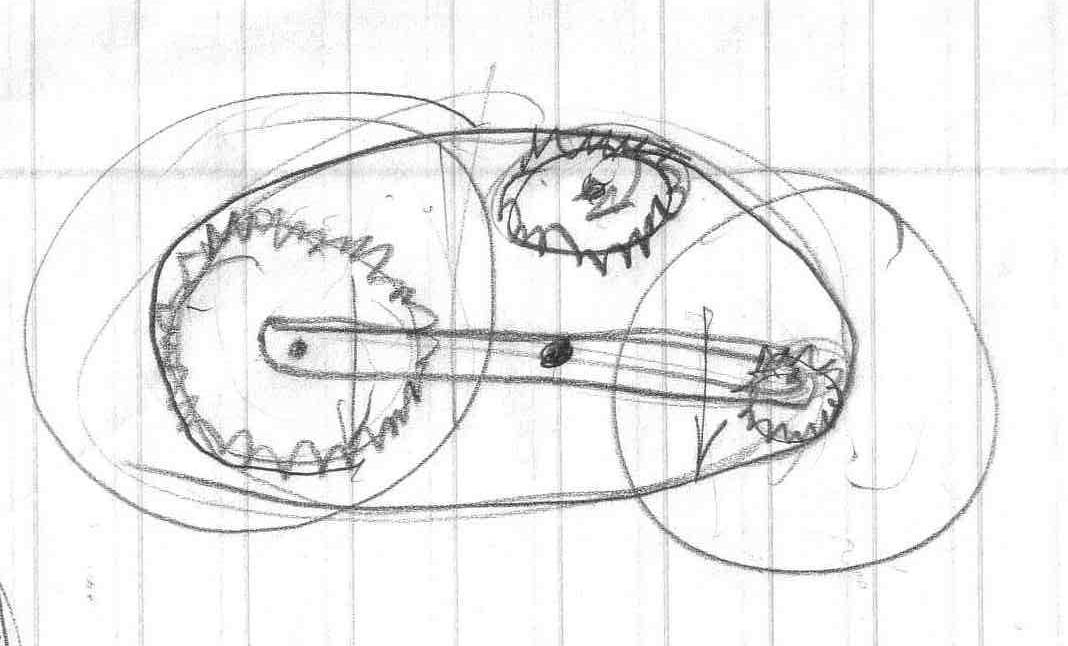
Working on the robot- switching tail wheel from servo to a DC motor

**Plastic cut diagrams:**

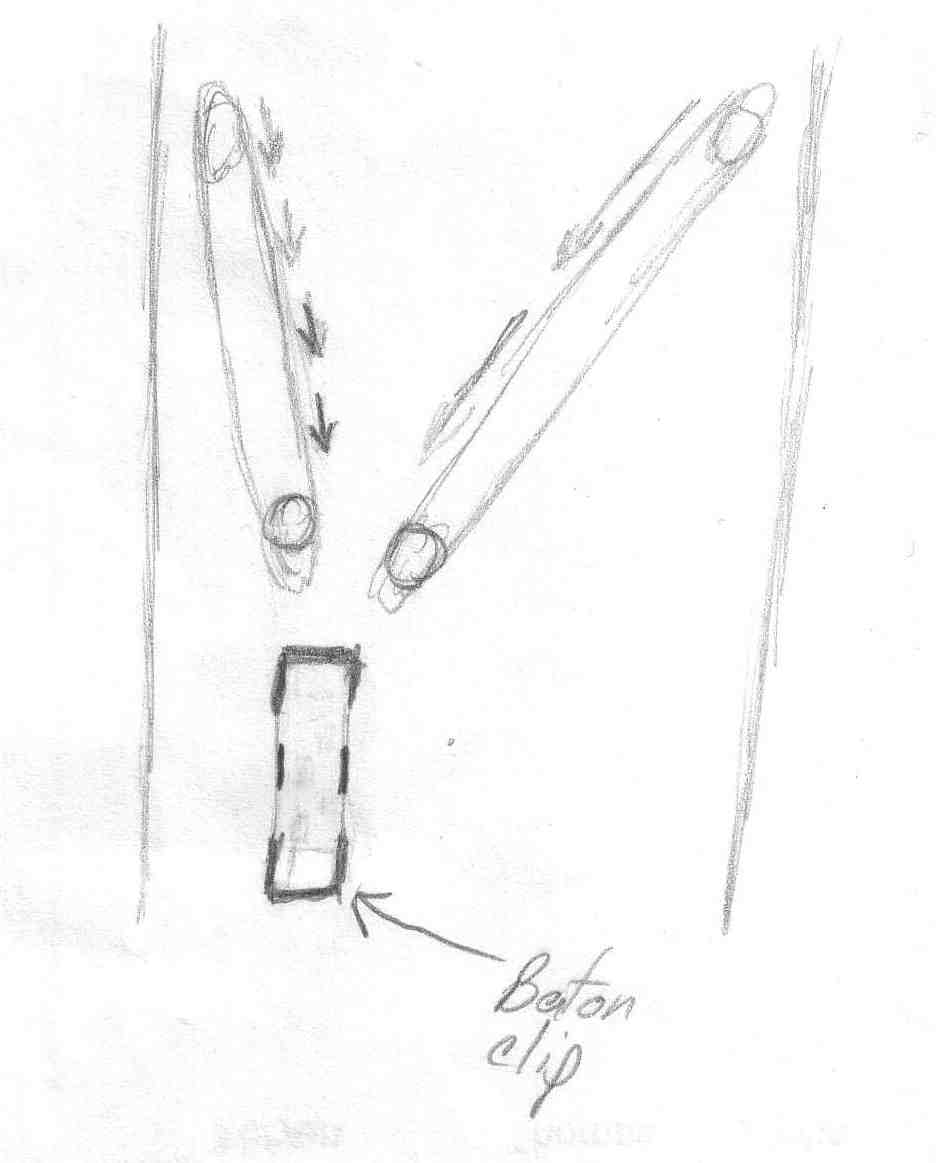




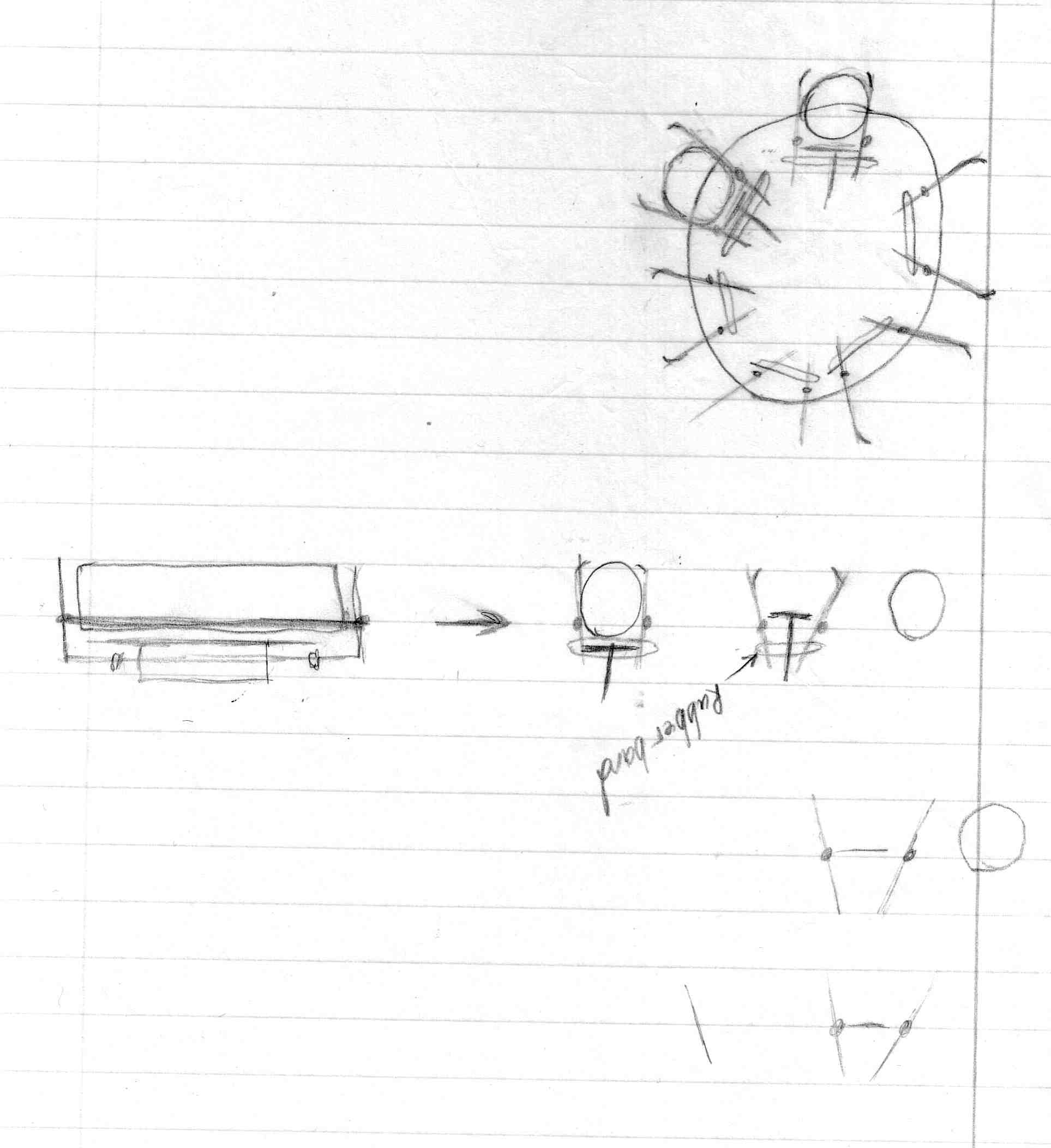
**Idea sketches:**



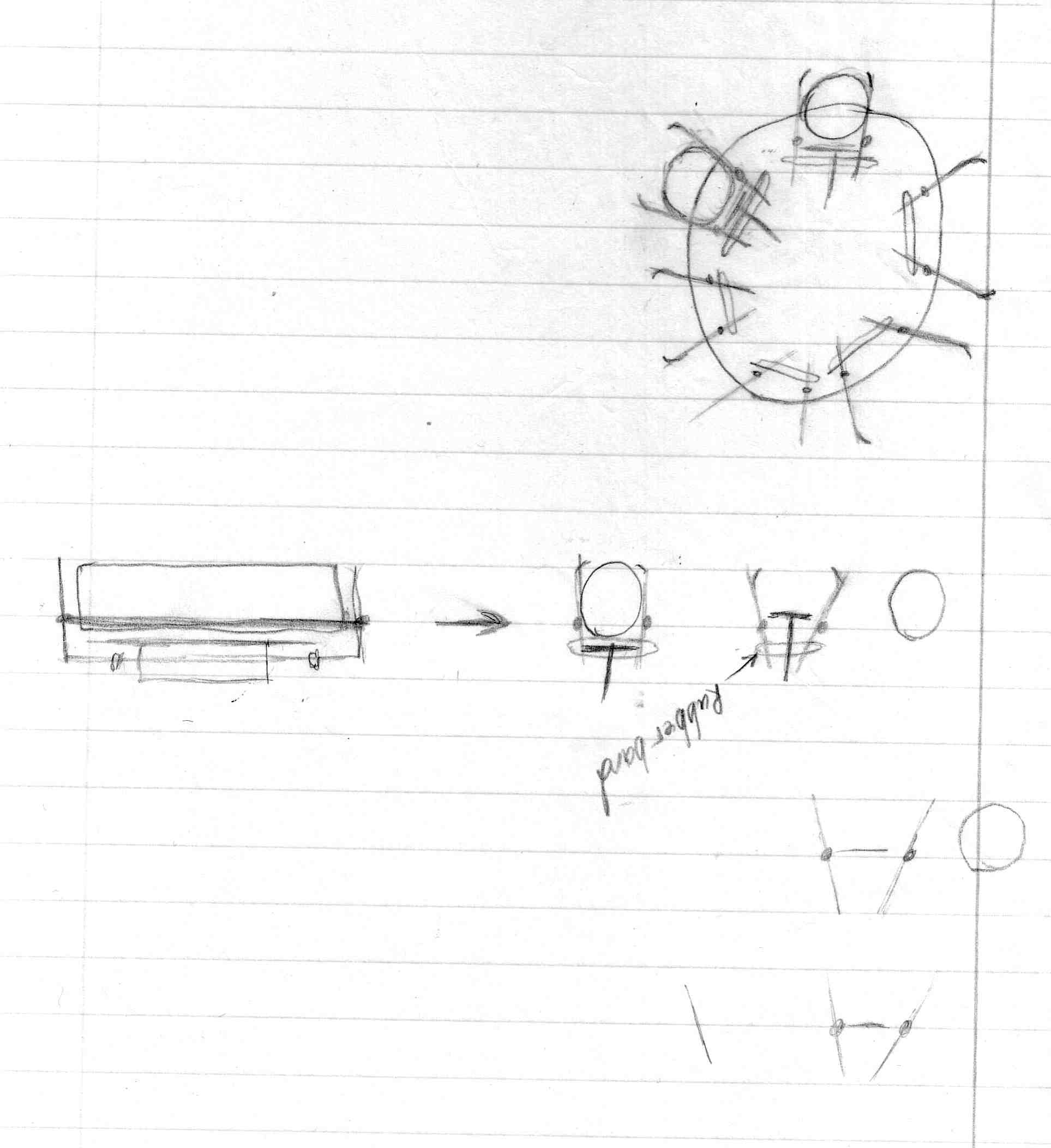
Gear shift diagram. We built a prototype but decided the effort to finish it would not provide many benefits for the risks involved.



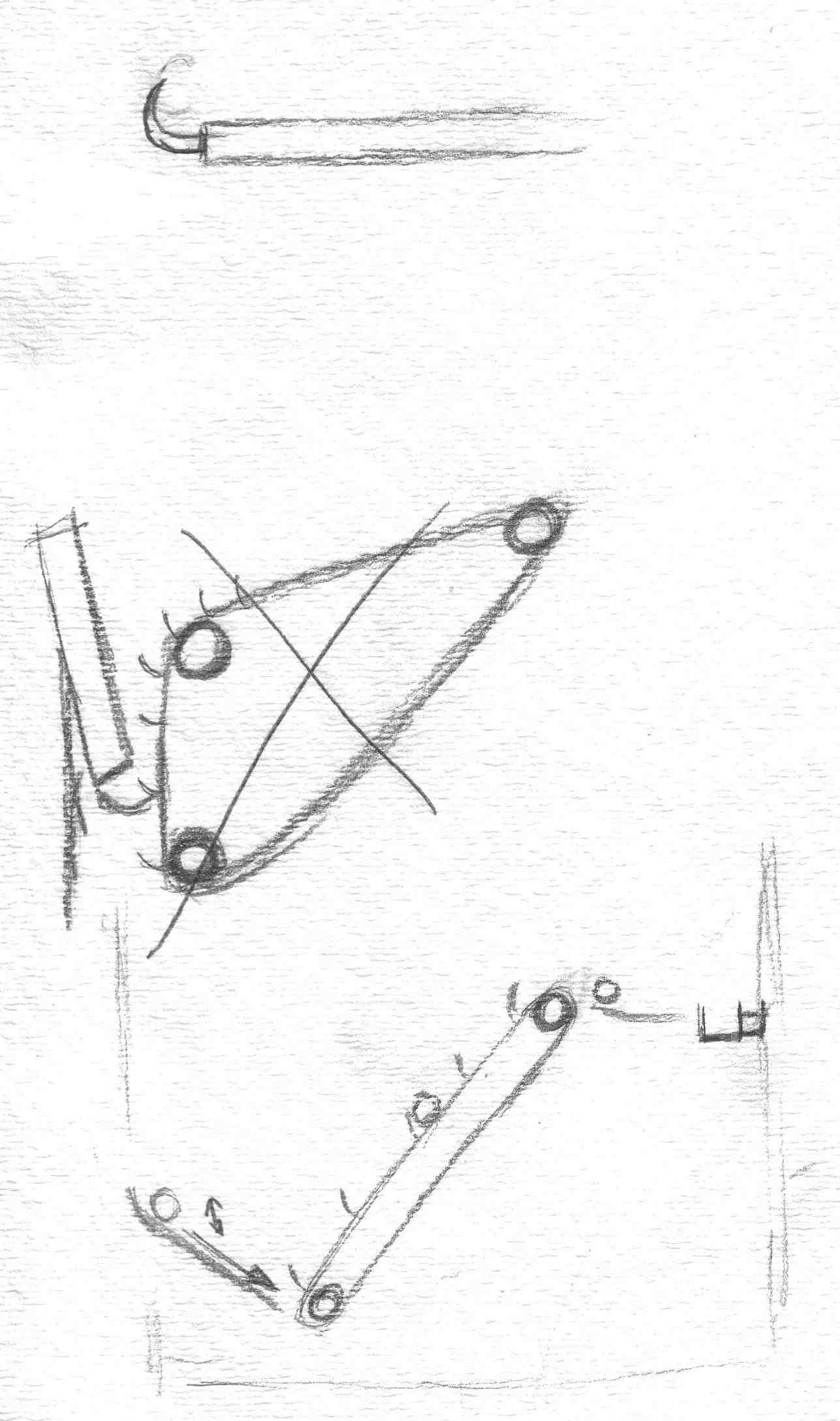
Baton clip with front V channel. Would hold five batons.



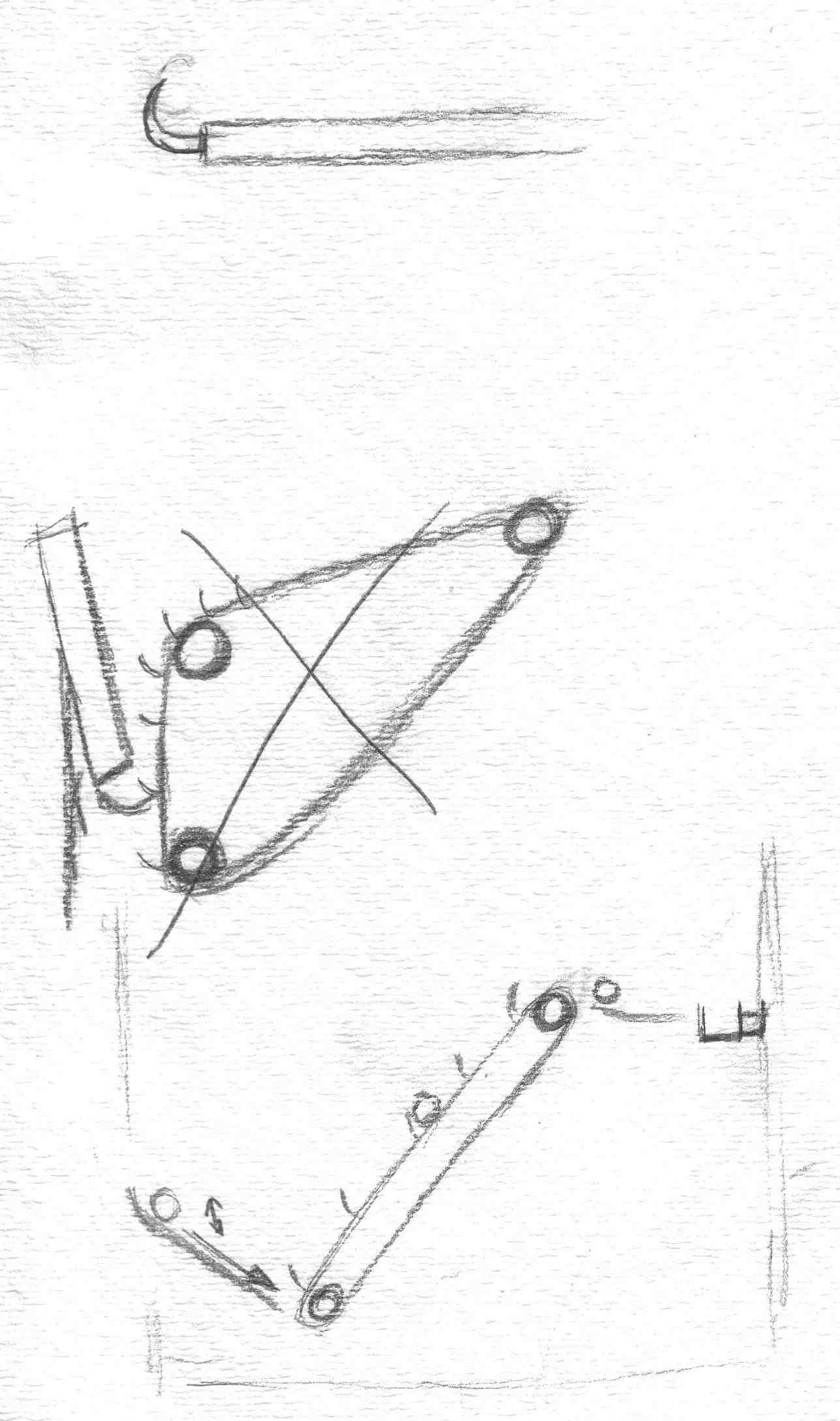
Baton holder idea



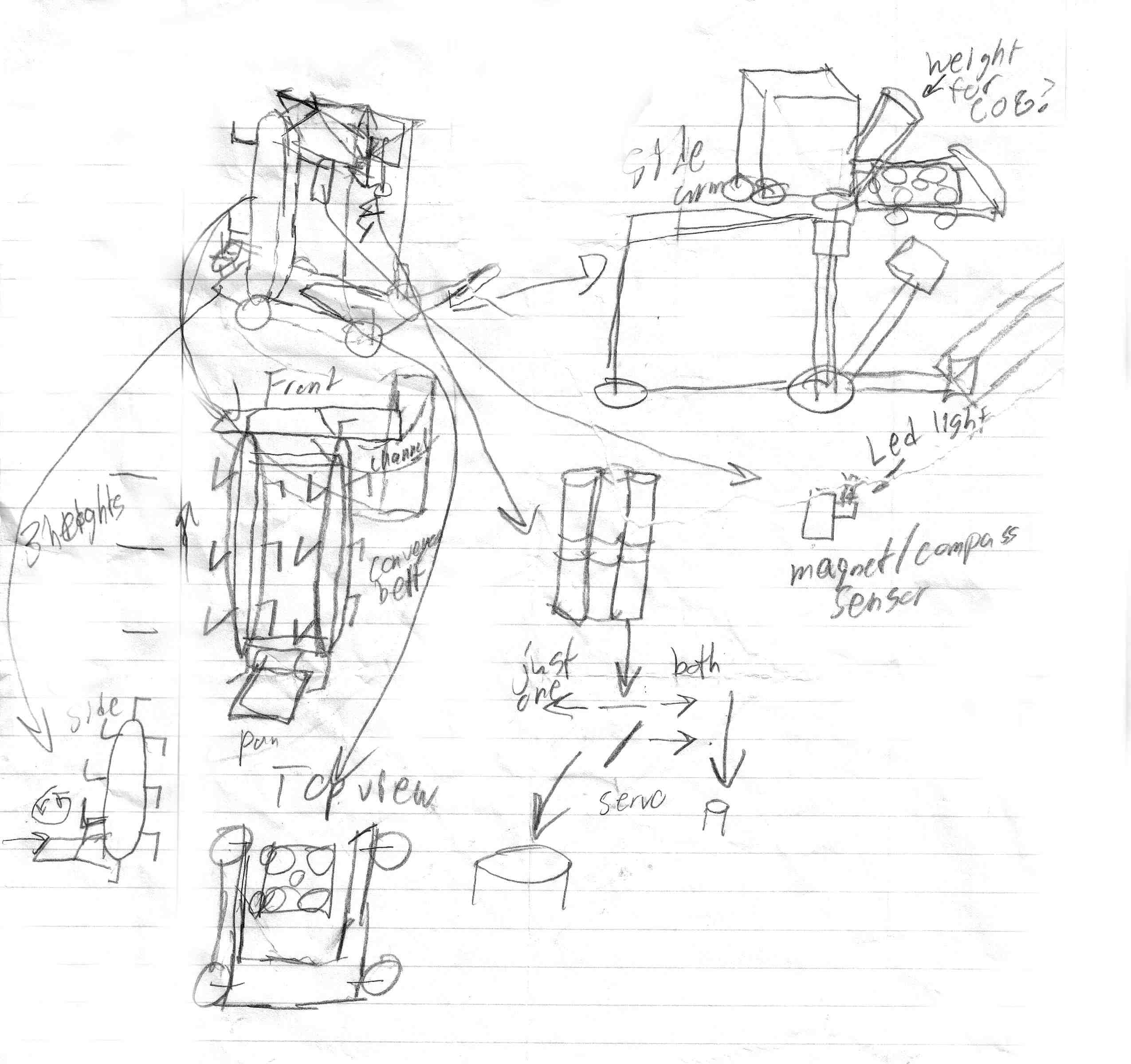
Clip and rubber band restraint



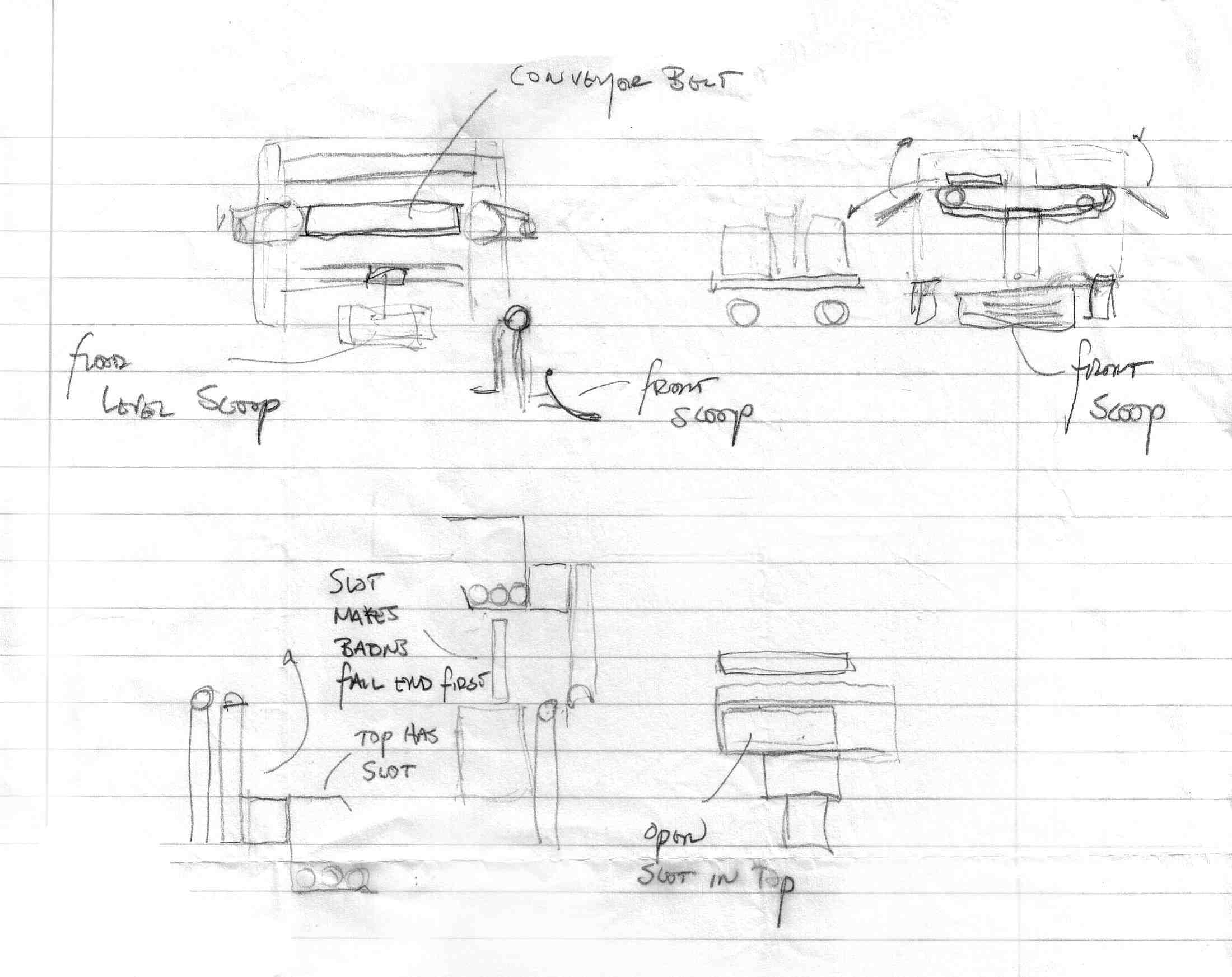
Cycling hook arm to possible be used for baton dispensing



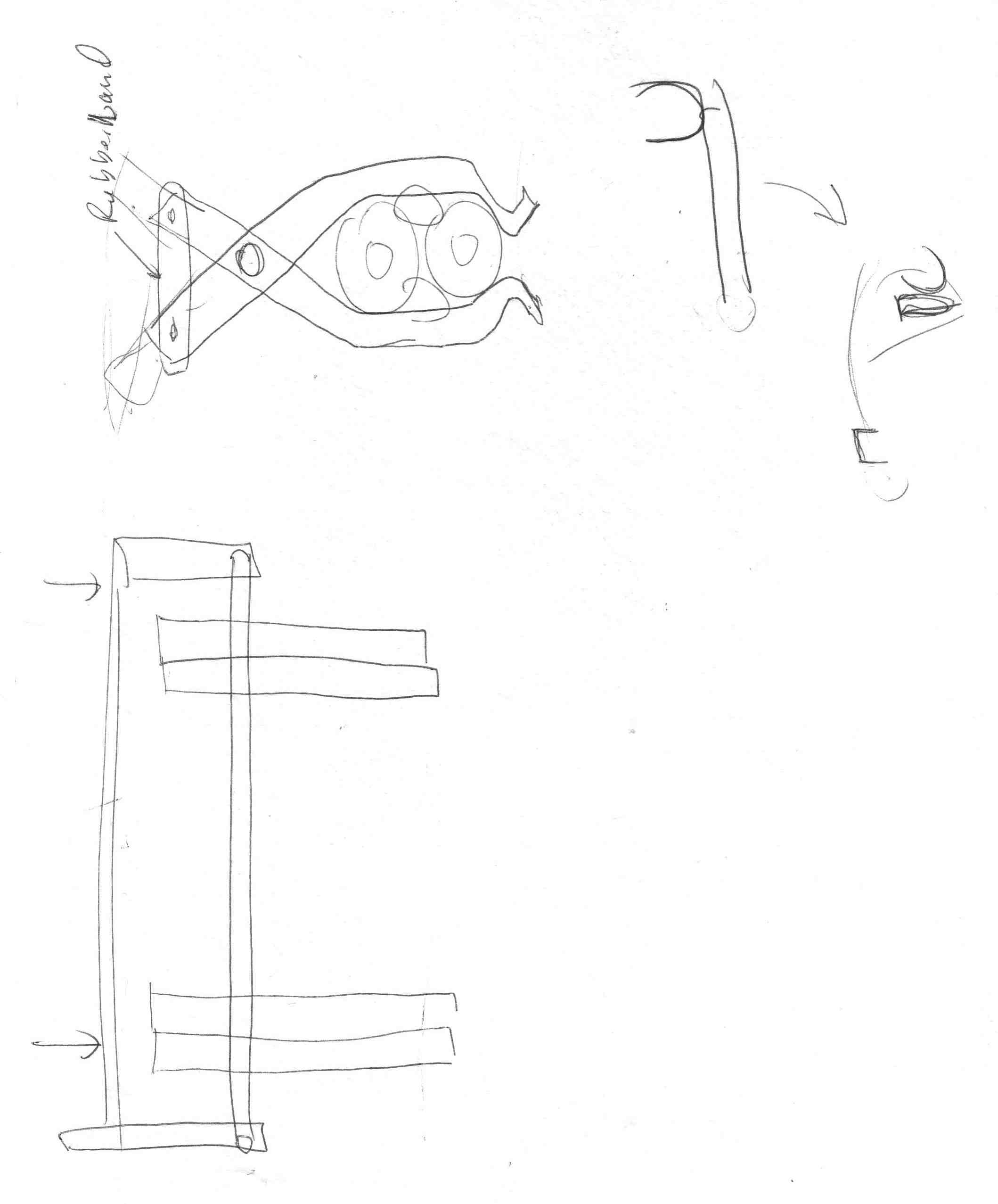
Conveyor belt to be used to pick up batons directly from dispenser and the floor.



Above: Entire robot design that a team member came up with. Contains ideas for baton conveyor belt, rolling goal corralling and containing. Gravity feed and straight down drop dispensing.



Baton intake ideas



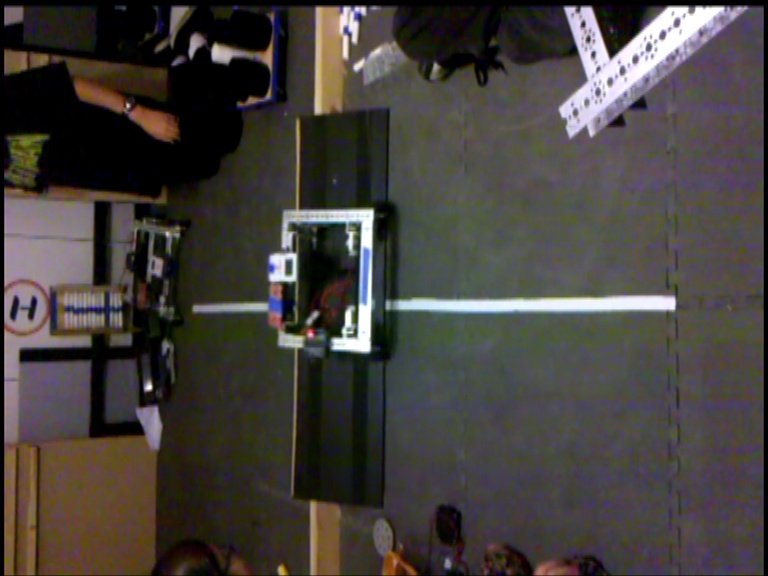
Sketch of claw for picking up batons from the ground. Would use pressure against the floor to over com force of rubber bands that would then hold the batons in.

**Multiple early chassis**:

The first chassis we built took a long time to make, but it worked very well for crossing the bridge. It was rectangular with two wheel drive. We wanted to modify it because we found we built it with more clearance than we actually needed. The next prototype wheel-base had less clearance but it turned out that it was out of dimension. The following chassis took advantage of field measurements to minimize the robot height while still clearing the mountain with room to spare.



First prototype chassis flies over the mountain. Built with more clearance than needed. We were surprised it was not that difficult to cross the mountain.



Second prototype takes the mountain at a more reasonable speed. It was too wide and occasionally tipped over. Needed to be redesigned.

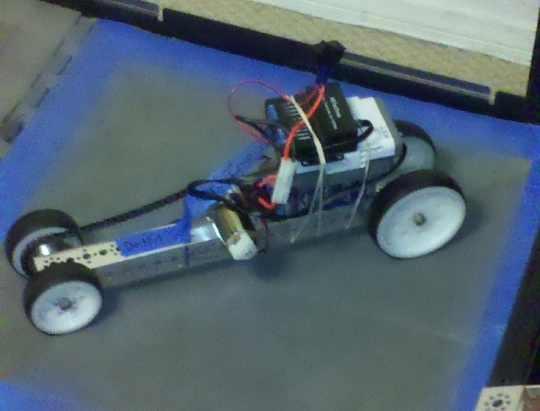


Third prototype attempted to fix the errors in the previous prototypes but still needed some modifications. We were a little bit too wide with this model.



This is the wheel-base that our team chose. It has two parallel beams that connect to another pair of slim metal beams. The wheels are attached to short pieces of U-channel at 45 degree angles. It was built with an optimum mixture of clearance, wide wheel spread and low center of gravity. The robot pictured here only has two motor drive. We currently use four motor drive at a one-to-one gear ratio.

**“Dragster”**



The prototype nicknamed the dragster was really the first test dispenser arm. It used a pair of wheels and a motor to speedily spill a dispenser of batons. Jokingly, a second pair of wheels was added to the metal beam so the arm could drive across the floor.

By using this arm we could remove all fifteen batons from a dispenser in about 2 seconds. It was mounted to the back end of the robot. It could also be used to hit the bridge down.

The current robot does not use a motorized wheel arm to dispense with because the latest hopper-scoop can dispense as easily as the old wheel arm could.

**Wiring Diagram:**

Right Back

Front of robot, hat

Right Front

Brush

Motor Controller 3

Motor 1 Motor 2

Servo Controller

Servos 1,2,3,4,5, 6

Left Back

Left Front

Motor Controller 2

Motor 1 Motor 2

Hopper

Motor Controller 1

Motor 1 Motor 2

2. Left Tail

1. Right Tail

**New Hopper**

5. Bridge Arm1

4. Hopper Flap

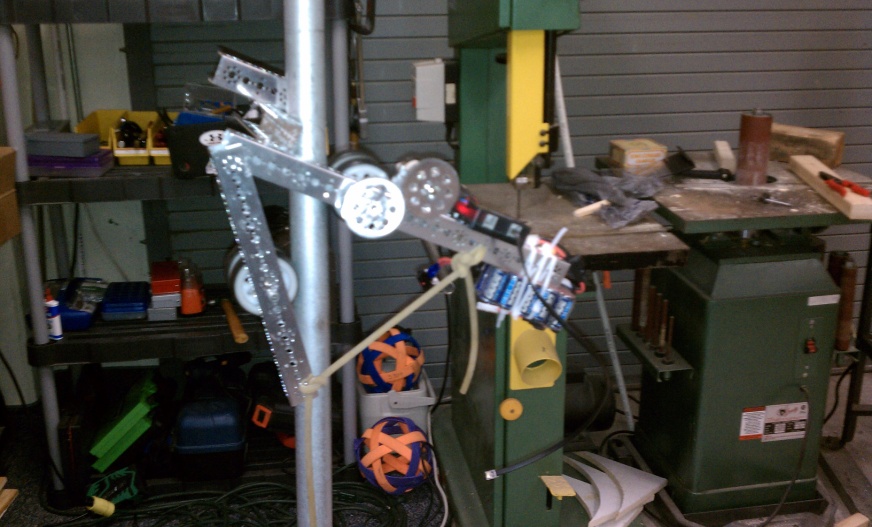
6. BridgeAm2

3. Preload

The new hopper combines features from the previous hopper with ones from the dispensing arm. This allows us to dispense, pick up, store, and score batons using just the hopper. It works better than the last one, although it is now heavier. We can pick up off the field and dispense from the same side of the baton. The hopper has a shield to prevent batons from falling out of it while crossing the mountain or other obstacles. The hopper uses a brush to get the batons in, and on the other end, a servo-powered flap covers the hole on the other end from which the batons all exit. When it’s correctly aligned, we can score all the batons in the hopper, and it is fast at picking up as well.

**AV High School Pole Robot**

The FRC teams this year are tasked with creating a mini-bot that can attach to a ten-foot metal pole and climb it in less than ten seconds. PHI volunteered to help a local team learn more about Tetrix parts and help them build their bot. We developed a functional prototype, which is capable of climbing the pole in about eight seconds. It was fun working with a FRC team and seeing another facet of high school FIRST robotics.

****

**New arms**

After the Las Vegas regional tournament, our team knew there were more changes that needed to be made for the robot’s improvement. After spending some time deciding what features were to be added, we put three more arms on to our robot.

Two arms serve to grab the rolling goal for end game and can be used earlier in the game as well. According to the rules, we cannot possess the rolling goal on all four sides until end game so we have programmed the arms to keep only three sides surrounded (not possess the goal) and still keep the goal with us. The other arm is used to quickly and simply score all five batons in autonomous mode.

Later we added yet another two arms in the front of the bot mounted with Lego motors. The triangle plastic pieces flip forward and are used to push the bridge down so we can drive onto it easily.

Our robot has thus been renamed the Kraken because of the number of the arms on the bot.

**Software:**

**Programming language choosing:**

Early in the season, we discussed what programming language to use, as there are two that we are allowed to use. First is Labview, a powerful visual-based program. Second is RobotC, a powerful word-based program. To choose, we talked about the pros and cons of both programs.

**Labview:** visual-based program

**Pros**: Its visual nature and creativity-stimulating layout. It is also easy to learn and good to use for gathering data and plotting it.

**Cons**: It is limited to the small number of programs that can fit in the NXT.

**RobotC:** word-based program

**Pros**: It is very logical. Our mentors have had experience in RobotC, and the programs take little room on the NXT.

**Cons**: It is hard to learn.

**Sensor Implementation:**

We are using nine sensors: five encoders, one ultrasonic sensor, one gyroscopic sensor, and an IR sensor.

**Encoders**: An encoder is a sensor that is used to sense the rotation of an axle.

**How we use it**: We use encoders for two reasons: to tell the robot how far to move and to tell the arms how far to rotate.

**IR Seeker: T**he IR Seeker does two things: find the IR emitter and tell what direction it is.

**How we use it:** We use the IR Seeker in a different way than most teams: to tell if there is something (like a robot) in front of the middle hopper.

**Gyro:** The gyroscopic sensor that lets us accurately measure the rotation of our robot. The gyro returns the number of degrees per second of rotation as well as the direction of rotation.

**How we use it:** We use the gyro to tell us how far we have turned.

**Ultrasonic**: The ultrasonic tells us how far away the closest object is.

**How we use it:** We use the ultrasonic sensor to tell us how far away the dispenser is.

**Pragmas:**

The pragmas are at the very beginning of the program to tell the programs where everything is plugged in to.the robot. They also allow you to name the motors and sensors to help you know what is what.

#pragma config(Hubs, S1, HTMotor, HTMotor, HTMotor, HTServo)

#pragma config(Sensor, S2, SonarLeft, sensorSONAR)

#pragma config(Sensor, S3, Gyro, sensorI2CHiTechnicGyro)

#pragma config(Sensor, S4, IR, sensorHiTechnicIRSeeker1200)

#pragma config(Motor, motorC, shield, tmotorNormal, openLoop)

#pragma config(Motor, mtr\_S1\_C1\_1, hopper, tmotorNormal, openLoop, reversed, encoder)

#pragma config(Motor, mtr\_S1\_C1\_2, brush, tmotorNormal, PIDControl, encoder)

#pragma config(Motor, mtr\_S1\_C2\_1, right\_front, tmotorNormal, PIDControl, reversed, encoder)

#pragma config(Motor, mtr\_S1\_C2\_2, right\_back, tmotorNormal, PIDControl, reversed, encoder)

#pragma config(Motor, mtr\_S1\_C3\_1, left\_front, tmotorNormal, PIDControl, encoder)

#pragma config(Motor, mtr\_S1\_C3\_2, left\_back, tmotorNormal, PIDControl, encoder)

#pragma config(Servo, srvo\_S1\_C4\_1, right\_tail, tServoStandard)

#pragma config(Servo, srvo\_S1\_C4\_2, left\_tail, tServoStandard)

#pragma config(Servo, srvo\_S1\_C4\_3, preload, tServoStandard)

#pragma config(Servo, srvo\_S1\_C4\_4, flap, tServoStandard)

#pragma config(Servo, srvo\_S1\_C4\_5, Bridge\_L, tServoStandard)

#pragma config(Servo, srvo\_S1\_C4\_6, Bridge\_R, tServoStandard)

**Includes/Functions/drivers:**

The includes tell the program where to find the functions on the computer. Functions are small parts of the program that are premade so we do not have to retype them over and over.

#include "Drivers/common.h"

#include "Drivers/HTGYRO-driver.h"

#include "Drivers/LEGOTS-driver.h"

#include "Drivers/HTIRS2-driver.h"

#include "Drivers/turn2\_gyro.c"

#include "Drivers/move.c"

#include "Drivers/score.c"

#include "Drivers/IR\_align.c"

#include "JoystickDriver.c"

**InitializeRobot**

The third part of a program is the initializeRobot. It is part of the templates that FIRST gave us that allows us to move or start motors before the match starts as long as we stay in the 18 by 18 by18 cube. We use it to hold the preloads in our hopper and to make sure the servo arms are in the right spot.

void initializeRobot()

{

servo[flap] = 0;

motor[shield] = 50;

servo[right\_tail] = 255;

servo[left\_tail] = 255;

servo[preload] = 145;

servo[Bridge\_L] = BL\_up;

servo[Bridge\_R] = BR\_up;

return;

}

**Tele-Op:**

In the tele-op program, there are four tasks that we run, but we only use three at one time. The first task is the “main task,” and it is used to run the other tasks and the buttons that run the many functions on the arms. The second task is the “drive task,” and it is used to run the wheels so that we move. Third is “DriveII” which is used only to cut the speed on the drive motor to slow down. The last is called failsafe which I will explain at the end of tele-op. We only use three tasks at the same time because that is the limit of the NXT.

task main()

{

if(timevoid == 0)

{

initializeRobot();

}

waitForStart();

if(timevoid == 0)

{

StartTask(failsafe);

//wait for start of tele-op phase

StartTask(drive);

timevoid = 1;

}

int Botton1 = 0;

int hopper\_position;

//int Tail\_State = 0;

int hopper\_pow = 0;

nMotorPIDSpeedCtrl[brush] = mtrSpeedReg;

nMaxRegulatedSpeed12V = 1000;

while(true)

{

if(joy1Btn(12))

{

StopTask(drive);

StartTask(driveII);

}

if(joy1Btn(11))

{

StopTask(driveII);

StartTask(drive);

}

if(joy1Btn(5))

{

servo[right\_tail] = 255;

servo[left\_tail] =255;

}

if(joy1Btn(7))

{

servo[left\_tail] = 0;

servo[right\_tail] = 0;

}

if(joy1Btn(6))

{

while(joy1Btn(6))

{

motor[hopper] = 50;//move the arm with the hopper up

}

motor(hopper) = 0;

wait10Msec(50);

hopper\_position = nMotorEncoder[hopper];

}

if(joy1Btn(8))

{

while(joy1Btn(8))

{

motor[hopper] = -40;//move the arm with the hopper up

//move the arm with the hopper down

}

motor(hopper) = 0;

hopper\_position = nMotorEncoder[hopper];

}

if(joy1Btn(2))

{

motor(brush) = 75;

wait10Msec(10);

motor(shield) = -75;

wait10Msec(50);

servo[flap] = 255;

wait10Msec(100);

motor( brush) = -30;

wait10Msec(10);

motor(shield) = 75;

wait10Msec(50);

motor(brush) = 0;

wait10Msec(10);

motor(hopper) = 80;

wait10Msec(30);

motor(hopper) = 0;

hopper\_position = nMotorEncoder[hopper];

servo[flap] = 00;

wait10Msec(10);

Botton1 = 0;

}

if(joy1Btn(3))

{

motor( brush) = 75;

wait10Msec(10);

motor(shield) = -75;

wait10Msec(50);

motor( brush) = -15;

wait10Msec(10);

Botton1 = 0;

}

if(joy1Btn(1))

{

motor( brush) = 75;

wait10Msec(10);

motor(shield) = -75;

wait10Msec(10);

motor(hopper) = -20;

wait10Msec(10);

Botton1 = 1;

}

if(joy1Btn(4))

{

motor( brush) = -75;

wait10Msec(10);

motor(shield) = 75;

wait10Msec(50);

motor(brush) = 0;

wait10Msec(10);

motor(hopper) = 0;

wait10Msec(10);

Botton1 = 0;

}

while (joystick.joy1\_TopHat == 3 || joystick.joy1\_TopHat == 4 || joystick.joy1\_TopHat == 5 )

{

servo[Bridge\_L] = BL\_down;

servo[Bridge\_R] = BR\_down;

}

while (joystick.joy1\_TopHat == 7 || joystick.joy1\_TopHat == 0 || joystick.joy1\_TopHat == 1 )

{

servo[Bridge\_L] = BL\_up;

servo[Bridge\_R] = BR\_up;

}

// nxtDisplayTextLine(3, "%d", nMotorEncoder[hopper]);

if( (nMotorEncoder[hopper] - hopper\_position) < -5 && Botton1 != 1 )

{

hopper\_pow = 5;

motor(hopper) = hopper\_pow;

}

else if( (nMotorEncoder[hopper] - hopper\_position) > 5 && Botton1 != 1 )

{

hopper\_pow = -5;

motor(hopper) = hopper\_pow;

}

else

{

hopper\_pow = 0;

motor(hopper) = hopper\_pow;

}

}

}

task drive()

{

nMotorPIDSpeedCtrl[left\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[right\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[right\_back] = mtrSpeedReg;

nMaxRegulatedSpeed12V = 2000;

int DeadBand = 15;

while(true) // Infinite loop:

{

getJoystickSettings(joystick);

if(abs(joystick.joy1\_y2) > DeadBand) // If the right analog stick's Y-axis readings are either above or below the threshold:

{

motor[right\_front] = joystick.joy1\_y2;

motor[right\_back] = joystick.joy1\_y2; // Motor D is assigned a power level equal to the right analog stick's Y-axis reading.

}

else // Else if the readings are within the threshold:

{

motor[right\_front] = 0;

motor[right\_back] = 0; // Motor D is stopped with a power level of 0.

}

if(abs(joystick.joy1\_y1) > DeadBand) // If the left analog stick's Y-axis readings are either above or below the threshold:

{

motor[left\_back] = joystick.joy1\_y1;

motor[left\_front] = joystick.joy1\_y1; // Motor E is assigned a power level equal to the left analog stick's Y-axis reading.

}

else // Else if the readings are within the threshold:

{

motor[left\_front] = 0;

motor[left\_back] = 0; // Motor E is stopped with a power level of 0.

}

}

}

task driveII

{

nMotorPIDSpeedCtrl[left\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[right\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[right\_back] = mtrSpeedReg;

nMaxRegulatedSpeed12V = 1000;

int DeadBand = 15;

while(true) // Infinite loop:

{

getJoystickSettings(joystick);

if(abs(joystick.joy1\_y2) > DeadBand) // If the right analog stick's Y-axis readings are either above or below the threshold:

{

motor[right\_front] = joystick.joy1\_y2/2;

motor[right\_back] = joystick.joy1\_y2/2; // Motor D is assigned a power level equal to the right analog stick's Y-axis reading.

}

else // Else if the readings are within the threshold:

{

motor[right\_front] = 0;

motor[right\_back] = 0; // Motor D is stopped with a power level of 0.

}

if(abs(joystick.joy1\_y1) > DeadBand) // If the left analog stick's Y-axis readings are either above or below the threshold:

{

motor[left\_back] = joystick.joy1\_y1/2;

motor[left\_front] = joystick.joy1\_y1/2; // Motor E is assigned a power level equal to the left analog stick's Y-axis reading.

}

else

{

motor[left\_front] = 0;

motor[left\_back] = 0;

}

}

}

task failsafe

{

long failmessage;

while(true)

{

failmessage = ntotalMessageCount;

wait10Msec(25);

if(failmessage == ntotalMessageCount)

{

motor[mtr\_S1\_C1\_1] = 0; // looking from the top, left side moves back, right

motor[mtr\_S1\_C1\_2] = 0; // side moves forward for CCW rotation

motor[mtr\_S1\_C2\_1] = 0;

motor[mtr\_S1\_C2\_2] = 0;

motor[mtr\_S1\_C3\_1] = 0;

motor[mtr\_S1\_C3\_2] = 0;

StopTask(drive);

StopTask(driveII);

StopTask(main);

nxtDisplayTextLine(2, "DISCONNECTED");

wait1Msec(10);

StartTask(main);

}

else

{

nxtDisplayTextLine(2, "CONNECTED");

}

}

}

**Tele-Op Controls**

****

Move RG arms up

Moves hopper up

Move RG arms down

Moves hopper down

Stop hopper movements

Move Bridge arms up

Picks up form ground

Move bridge arms down

Dispense into hopper

Reless Batons in bot

Contrals left side of bot

And if pressed switches form diveII to dive

Contrals right side of bot

And if pressed switches from dive to driveII

**Watchdog/failsafe**

The watchdog program or function is something we created to protect our bot if we lose connection. Normally if we lose connection with RobotC, our robot will do the last command repeatedly till the battery dies or someone stops it. This program can tell if the bot is connected to the FCS by watching a value that is in the Joystick Driver. The value is constantly going up as long as the bot is connected. If the value stops going up, the program stops the motors and the tasks that run the bot, stopping the bot. If it regains connection, the bot restarts the tasks. This is the code are giving to every FTC team we can.

////////////////////////////////////////////////////////////////////////////////////////

///

/// What it is: the watchdog program is a subroutine that is put in a teleop program

/// to watch the connection from the Bluetooth or Samantha and stop the motors if necessary.

///

/// License: You may use or modify this code as you wish, provided you give credit to PHI 4322.

///

/// Installation instructions:

/// 1. Insert the names of the tasks that are in your teleop into the the spots named "StartTask" and "StopTask."

/// 2. Copy all the code from the WatchDog task and paste it at the very end of your teleop program.

/// 3. Before the WaitForStart in task main, type "StartTask(WatchDog);"

/// 4. Below the "#include JoystickDriver.c" type "task failsafe();"

/// 5. Then click F7 to compile; if you have errors in the task, repeat the instructions and try again.

///

/////////////////////////////////////////////////////////////////////////////////////////

task WatchDog

{

float watchedmessage;

while(true)

{

watchedmessage = ntotalMessageCount; // grabs the number of Messages and names it watchedMessage

wait10Msec(15); // give the Messagecount time to change it changes every 50ms

if(watchedmessage == ntotalMessageCount) // checks to see if the Message change if has not it is Disconnected

{ // if Disconnected it runs this

motor[mtr\_S1\_C1\_1] = 0; // stops motor

motor[mtr\_S1\_C1\_2] = 0;

motor[mtr\_S1\_C2\_1] = 0;

motor[mtr\_S1\_C2\_2] = 0;

motor[mtr\_S1\_C3\_1] = 0;

motor[mtr\_S1\_C3\_2] = 0;

motor[mtr\_S1\_C4\_1] = 0;

motor[mtr\_S1\_C4\_2] = 0;

StopTask("insert name of task"); //stops task robot control task

StopTask("insert name of task");

wait1Msec(10); //gives time to spot

StartTask("insert name of task"); //restart task incase of reconnection

StartTask("insert name of task");

nxtDisplayTextLine(2, "DISCONNECTED"); //displays DISCONNECTED to tell you if you are Disconnected

}

else//if the Message change you are connected

{

nxtDisplayTextLine(2, "CONNECTED"); //displays CONNECTED to tell you if everything is working

}

}

}

**Autonomous code:**

There are three main parts of the autonomous codes. The first part is the functions- the subroutines that contain the small details to make the robot move. The second part is the main part of the program, the part in which the user tells the robot what to do.

**Function: Move**

Move is the first function we use. It is a function that tells the robot to drive forward and backward. We can also tell the robot what power level the motors should run at.

pragma platform(FTC) // Only works for "FTC" platform. Generate error on other platforms.

#pragma systemFile // This eliminates warning for "unreferenced" functions

// Fastest the bot will move with nMotorEncoderTarget[]

// Speed bot moves during sprints

#define COUNTS\_PER\_INCH 112 // Encoder counts per inch

//========================================================================

// Function: move()

//

// Description:

// This function will move the robot forward or backward the amount of inches

// specified in "distance".

//

// Parameters: distance = inches to move, Positive is forward,

// Negative is backward

//int move( float distance , float MAX\_SPEED, long offset )

int move( float distance , float MAX\_SPEED, int gyro\_offset)

{

int CountsToGo;

int CountDistance;

int AvgMotorEncoder;

int DiffMotorEncoder;

int Direction = 1;

int MotorSpeed = 0;

int SlowPoint;

int timemax;

int GyroReading1, GyroReading2;

int Delta\_left\_front;

int Delta\_right\_front;

int Delta\_left\_back;

int Delta\_right\_back;

int start\_left\_front;

int start\_right\_front;

int start\_left\_back;

int start\_right\_back;

int last\_left\_front;

int last\_right\_front;

int last\_left\_back;

int last\_right\_back;

long deg = 0;

long Start\_time;

long Delta\_time;

long End\_time;

long Rotation\_rate=0;

ClearTimer(T1);

// Determine direction. Set to negative for backwards

if( distance < 0 )

{

Direction = -1;

}

// Calculate the distance to travel in encoder counts

CountDistance = abs(distance) \* COUNTS\_PER\_INCH;

//CountDistance = 1000;

// remove the backlash and freeplay from the motors before zeroing the encoders by commanding

// a very low power for a short time

// reset the front motor encoders to zero

nMotorEncoder[right\_front] = 0;

nMotorEncoder[left\_front] = 0;

nMotorEncoder[right\_back] = 0;

nMotorEncoder[left\_back] = 0;

if(distance > 5)

{

MotorSpeed = 5;

}

else

{

MotorSpeed = MAX\_SPEED;

}

motor[left\_front] = Direction \* MotorSpeed;

motor[right\_front] = Direction \* MotorSpeed;

motor[left\_back] = Direction \* MotorSpeed;

motor[right\_back] = Direction \* MotorSpeed;

wait1Msec(25);

// MotorSpeed = MAX\_SPEED;

// Now ramp the motor power at a fixed rate to provide a "soft start" to the drive motors.

// At 40 ms/increment, this will take approximately 680 ms to get the motors to full speed

// Do this for only for distances > 8 inches

if (abs(distance)> 5)

{

while (MotorSpeed <= MAX\_SPEED)

{

MotorSpeed = MotorSpeed + 5;

motor[left\_front] = Direction \* MotorSpeed;

motor[right\_front] = Direction \* MotorSpeed;

motor[left\_back] = Direction \* MotorSpeed;

motor[right\_back] = Direction \* MotorSpeed;

wait1Msec(100);

}

}

// motor[right\_front] = 0;

// motor[right\_back] = 0;

GyroReading1 = (SensorValue[Gyro]-gyro\_offset); // Gyro connected to NXT

ClearTimer(T2);

//---------------------------------------------------------------------------

// Move the motors until one of the encoders indicate the distance has been traveled

//---------------------------------------------------------------------------

while( ( abs(nMotorEncoder[left\_front]) < CountDistance ) &&

( abs(nMotorEncoder[right\_front]) < CountDistance ) )

{

End\_time = time1[T1];

Delta\_time = End\_time - Start\_time;

Start\_time = End\_time;

GyroReading2 = GyroReading1;

GyroReading1 = (SensorValue[Gyro]-gyro\_offset);

Rotation\_rate = (GyroReading1 + GyroReading2)/2;

deg = deg + (Delta\_time \* Rotation\_rate);

wait10Msec(15);

start\_left\_front = abs(nMotorEncoder[left\_front]);

start\_right\_front = abs(nMotorEncoder[right\_front]);

start\_left\_back = abs(nMotorEncoder[left\_back]);

start\_right\_back = abs(nMotorEncoder[right\_back]);

Delta\_left\_front = abs(last\_left\_front - start\_left\_front);

Delta\_right\_front = abs(last\_right\_front - start\_right\_front);

Delta\_left\_back = abs(last\_left\_back - start\_left\_back);

Delta\_right\_back = abs(last\_right\_back - start\_right\_back);

last\_left\_front = start\_left\_front;

last\_right\_front = start\_right\_front;

last\_left\_back = start\_left\_back;

last\_right\_back = start\_right\_back;

if (time100[T2] > 10000)

{

if (abs(deg) > 30\*1000 || Delta\_left\_front < 10 || Delta\_left\_back < 10 || Delta\_right\_front < 10 || Delta\_right\_back < 10 )

{

motor[right\_front] = 0;

motor[right\_back] = 0;

motor[left\_front] = 0;

motor[left\_back] = 0;

}

}

}

// End while distance remaining to travel

// since we have now reached the target distance (the while loop is done), stop all drive motors

motor[right\_front] = 0;

motor[right\_back] = 0;

motor[left\_front] = 0;

motor[left\_back] = 0;

wait10Msec(10);

deg = deg / 1000;

return deg;

}

**Gyro turn**

Turn is the second function we use. It tells the robot to spin clockwise or counterclockwise, using the gyro to tell how far to turn.

#pragma platform(FTC) // Only works for "FTC" platform. Generate error on other platforms.

#pragma systemFile // This eliminates warning for "unreferenced" functions

//=============================================================================

// Function: turn2\_gyro()

//

// Description:

// This function will turn the robot counter clock-wise (CCW) in place by

// the number of degrees specified in "turnDegrees". This function uses the gyro and

// integrates to determine the angle

//

// Parameters: turnDegrees = degrees to rotate, Positive is CCW,

// Negative is CW

//=============================================================================

void turn( float turnDegrees, float GyroBias )

{

int time;

int time\_old = 0;

int delta\_time;

float maxVelocity = 100.0; // max angular velocity in deg/sec

float GyroOld;

float GyroNew;

int rotDirection = 1;

int motorSpeed = 0;

float DegreeGain = 5.0;

float degreesToGo;

int MAX\_GYRO\_SPEED = 80; //maximum motor speed allowed in turns

int MIN\_GYRO\_SPEED = 15; //minimum motor speed allowed in turns

float Angle;

// Determine the rotation direction. Set to negative for clock-wise rotation.

if( turnDegrees < 0 ) rotDirection = -1;

// Set the Angle and components to zero befor turning

Angle = 0.0;

degreesToGo = 0.0;

time\_old = nPgmTime;

GyroOld = 0.0;

wait1Msec(100);

// remove the backlash and freeplay from the motors before zeroing the encoders by commanding

// a very low power for a short time

// reset the back motor encoders to zero

nMotorEncoder[right\_back] = 0;

nMotorEncoder[left\_back] = 0;

motorSpeed = 5;

motor[left\_front] = -rotDirection \* motorSpeed ; // looking from the top, left side moves back, right

motor[right\_front] = rotDirection \* motorSpeed ; // side moves forward for (+) CCW rotation

motor[left\_back] = -rotDirection \* motorSpeed ;

motor[right\_back] = rotDirection \* motorSpeed ;

wait1Msec(10);

ClearTimer(T3);

//---------------------------------------------------------------------------

// Move the motors until the gyro Angle indicates the rotation is complete

//---------------------------------------------------------------------------

while( ( abs(turnDegrees) - abs(Angle) ) > 3 )

{

time = nPgmTime; // this timer wraps around

delta\_time = abs(time - time\_old); // delta time in ms

if (delta\_time < 1) // protect against divide by zero

{

delta\_time = 1;

}

// read the gyro sensor minus the bias offset. GyroBias must be declared and

// computed in the calling program.

GyroNew = -((float)SensorValue[Gyro] - GyroBias);

// limit the gyro to the max achievable by the bot to minimize data spikes.

if (abs(GyroNew) > maxVelocity) GyroNew = sgn(GyroNew)\*maxVelocity;

// deadband for the gyro to eliminate drift due to noise

if (abs(GyroNew) <= 0.2) GyroNew = 0.0;

// compute the integral of the angular rate using a trapazoidal approximation

// http://en.wikipedia.org/wiki/Numerical\_integration

Angle = Angle + 0.001 \* (float)delta\_time \* 0.5 \*(GyroNew + GyroOld);

// update the old values for the next time through the loop

time\_old = time;

GyroOld = GyroNew;

// Calculate the rotation remaining

degreesToGo = abs(turnDegrees) - abs(Angle);

// motor speed is proportional to the amount of rotation remaining

motorSpeed = (int)(DegreeGain \* abs(degreesToGo));

// limit the motor speed to be greater than 15 and less than 75

if (abs(motorSpeed) > MAX\_GYRO\_SPEED) motorSpeed = MAX\_GYRO\_SPEED;

if (abs(motorSpeed) < MIN\_GYRO\_SPEED) motorSpeed = MIN\_GYRO\_SPEED;

motor[left\_front] = -rotDirection \* motorSpeed ; // looking from the top, left side moves back, right

motor[right\_front] = rotDirection \* motorSpeed ; // side moves forward for (+) CCW rotation

motor[left\_back] = -rotDirection \* motorSpeed ;

motor[right\_back] = rotDirection \* motorSpeed ;

if( time100[T3] > 50 ) break; // if the bot is not done turning in 5 sec, quit

wait1Msec(15); // wait 20 ms to allow a reasonable period for the integration

} // End while rotation remaining

// since we have now reached the target rotation (the while loop is done), stop all drive motors

motor[right\_front] = 0;

motor[right\_back] = 0;

motor[left\_front] = 0;

motor[left\_back] = 0;

// reset the back motor encoders to zero

nMotorEncoder[right\_back] = 0;

nMotorEncoder[left\_back] = 0;

}

**Arc\_Turn**

Arc turn is a new function and use to turn on a arc. Using PID we were able to control the speed need to turn on a arc. We use it to push a rolling goal into the Stationary goal or to push it over the Mountain.

#pragma platform(FTC) // Only works for "FTC" platform. Generate error on other platforms.

#pragma systemFile // This eliminates warning for "unreferenced" functions

//=============================================================================

// Function: Arc\_turn()

//

// Description:

// This function will turn the robot counter clock-wise (CCW) in place by the number of degrees

// specified in "turn\_degrees".

//

// Parameters: turn\_degrees = degrees to rotate, Positive is CCW,

// Negative is CW

//

// This subroutine uses the Hitech Gyro Sensor to measure the rotation rates.

// The individual ratatioal rates are multiplied by the delta time to determine the degrees rotated.

// These small degrees rotation are summed to compute the total number of degrees rotated.

//

//=============================================================================

void arc\_turn( long turn\_degrees, float MotorSpeed, float inner\_radius, int gyro\_offset)

{

long deg = 0;

long Start\_time;

long Delta\_time;

long End\_time;

long Rotation\_rate = 0;

int GyroReading1, GyroReading2;

float speed\_ratio = 0;

float wheel\_base = 16.5;

float radius\_adj = 0.625;

inner\_radius = inner\_radius \* radius\_adj;

speed\_ratio = ((wheel\_base + inner\_radius) / inner\_radius);

if (turn\_degrees > 0)

{

motor(left\_back) = MotorSpeed / speed\_ratio;

motor(left\_front) = MotorSpeed / speed\_ratio;

motor(right\_back) = MotorSpeed;

motor(right\_front) = MotorSpeed;

wait10Msec(5);

}

else

{

motor(left\_back) = MotorSpeed;

motor(left\_front) = MotorSpeed;

motor(right\_back) = MotorSpeed / speed\_ratio;

motor(right\_front) = MotorSpeed / speed\_ratio;

wait10Msec(5);

}

turn\_degrees = turn\_degrees\*1000;

ClearTimer(T1);

ClearTimer(T2);

// sets thew initial rotational rate

Start\_time = time1[T1];

GyroReading1 = abs(SensorValue[Gyro]-gyro\_offset); // Gyro connected to NXT

while(deg < abs(turn\_degrees)) // checks it bot has rotated the desired # of degrees

{

End\_time = time1[T1];

Delta\_time = End\_time - Start\_time;

Start\_time = End\_time;

GyroReading2 = GyroReading1;

GyroReading1 = abs(SensorValue[Gyro]-gyro\_offset);

Rotation\_rate = (GyroReading1 + GyroReading2)/2;

deg = deg + Delta\_time \* abs(Rotation\_rate);

wait10Msec(20);

if (time100[T2] > 50) break;

}

// stop all drive motors

motor[left\_front] = 0;

motor[right\_front] = 0;

motor[left\_back] = 0;

motor[right\_back] = 0;

wait10Msec(10);

}

**IR\_align**

IR align is a function that aligns the robot to the dispenser using the IR under it. We use IR align so we know that nothing is blocking the way and we can dispense the middle dispenser.

#pragma platform(FTC) // Only works for "FTC" platform. Generate error on other platforms.

#pragma systemFile // This eliminates warning for "unreferenced" functions

//=============================================================================

void IR\_align(int gyro\_offset)

{

int acS1, acS2, acS3, acS4, acS5 = 0;

int in, SL;

//long offset;

//long gyro\_offset = offset;

long deg = 0;

long Start\_time;

long Delta\_time;

long End\_time;

long Rotation\_rate=0;

int encoder\_right1, encoder\_left1;

int encoder\_right2, encoder\_left2;

int GyroReading1, GyroReading2;

SL = SensorValue[SonarLeft];

wait10Msec(10);

// SR = SensorValue[SonarRight];

// wait10Msec(10);

if( SL > 200 ) SL = 30 \* 2.54;

in = SL / 2.54;

HTIRS2readAllACStrength(msensor\_S4\_2, acS1, acS2, acS3, acS4, acS5 );

wait10Msec(10);

ClearTimer(T1);

//offset = HTGYROreadRot(msensor\_S4\_1);

// }

if (acS2 > 0)

{

motor[left\_front] = -30; // looking from the top, left side moves back, right

motor[right\_front] = 30; // side moves forward for CCW rotation

motor[left\_back] = -30;

motor[right\_back] = 30;

wait10Msec(10);

}

while(acS2 > 0)

{

End\_time = time1[T1];

Delta\_time = End\_time - Start\_time;

Start\_time = End\_time;

GyroReading2 = GyroReading1;

GyroReading1 = (SensorValue[Gyro]-gyro\_offset);

Rotation\_rate = (GyroReading1 + GyroReading2)/2;

deg = deg + (Delta\_time \* Rotation\_rate);

HTIRS2readAllACStrength(msensor\_S4\_2, acS1, acS2, acS3, acS4, acS5 );

wait10Msec(10);

}

motor[left\_front] = 0;

motor[right\_front] = 0;

motor[left\_back] = 0;

motor[right\_back] = 0;

wait10Msec(10);

GyroReading1 = (SensorValue[Gyro]-gyro\_offset); // Gyro connected to NXT

if (abs(acS3) > 0 || abs(acS4) > 0)

{

motor[left\_front] = 30;

motor[right\_front] = -30;

motor[left\_back] = 30;

motor[right\_back] = -30;

wait10Msec(10);

while (acS2<10)

{

End\_time = time1[T1];

Delta\_time = End\_time - Start\_time;

Start\_time = End\_time;

GyroReading2 = GyroReading1;

GyroReading1 = (SensorValue[Gyro]-gyro\_offset);

Rotation\_rate = (GyroReading1 + GyroReading2)/2;

deg = deg + (Delta\_time \* Rotation\_rate);

HTIRS2readAllACStrength(msensor\_S4\_2, acS1, acS2, acS3, acS4, acS5 );

} // end of while loop

}

else if (acS1 > 0)

{

motor[left\_front] = -30; // looking from the top, left side moves back, right

motor[right\_front] = 30; // side moves forward for CCW rotation

motor[left\_back] = -30;

motor[right\_back] = 30;

wait10Msec(10);

while (acS3<10)

{

End\_time = time1[T1];

Delta\_time = End\_time - Start\_time;

Start\_time = End\_time;

GyroReading2 = GyroReading1;

GyroReading1 = (SensorValue[Gyro]-gyro\_offset);

Rotation\_rate = (GyroReading1 + GyroReading2)/2;

deg = deg + (Delta\_time \* Rotation\_rate);

HTIRS2readAllACStrength(msensor\_S4\_2, acS1, acS2, acS3, acS4, acS5 );

} // end of while loop

}

motor[left\_front] = 0;

motor[right\_front] = 0;

motor[left\_back] = 0;

motor[right\_back] = 0;

wait10Msec(20);

motor(hopper) = -50;

wait10Msec(150);

motor(hopper) = 0;

wait10Msec(10);

deg = deg / 1000;

move((in-11), 30, gyro\_offset);

wait10Msec(5);

if (abs(deg) > 15)

{

turn(deg, gyro\_offset);

wait10Msec(5);

}

//time100[T4] = 0;

nMotorEncoder[left\_front] = 0;

nMotorEncoder[right\_front] = 0;

nMotorPIDSpeedCtrl[left\_front] = mtrNoReg;

nMotorPIDSpeedCtrl[right\_front] = mtrNoReg;

encoder\_left1 = nMotorEncoder[left\_front];

encoder\_right1 = nMotorEncoder[right\_front];

motor[left\_front] = 20;

motor[right\_front] = 20;

motor[left\_back] = 20;

motor[right\_back] = 20;

wait10Msec(50);

encoder\_left2 = nMotorEncoder[left\_front];

encoder\_right2 = nMotorEncoder[right\_front];

while (abs(encoder\_left1 - encoder\_left2) > 10 && abs(encoder\_right1 - encoder\_right2))

// while(time100[T4] < 5);

{

encoder\_left2 = encoder\_left1;

encoder\_right2 = encoder\_right1;

encoder\_left1 = nMotorEncoder[left\_front];

encoder\_right1 = nMotorEncoder[right\_front];

wait10Msec(25);

}

motor[left\_front] = 0;

motor[right\_front] = 0;

motor[left\_back] = 0;

motor[right\_back] = 0;

wait10Msec(5);

nMotorPIDSpeedCtrl[left\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[right\_front] = mtrSpeedReg;

wait10Msec(5);

}

**Score:**

Score is the last function, which we use to make our hopper scale up the wall to dispense a dispenser of any height.

#pragma platform(FTC) // Only works for "FTC" platform. Generate error on other platforms.

#pragma systemFile // This eliminates warning for "unreferenced" functions

//=============================================================================

// Function: shoot(int shots)

//

// Description:

// This function will shoot a given number of wiffle balls at a set power; number of shots specified in "shots."

//

// Parameters: shots = number of balls to fire, Positive and Negative will both produce forward rotation

//=============================================================================

void score(int ScoreTime, int wiggle)

{

int MaxAngle = 1200;

int EncoderReading, brush\_encoder;

int StartCount = 0;

int CountDistance = 672;

int hopper\_power = 80;

int drive\_power = -7;

motor(brush)= -100;

motor(hopper) = hopper\_power;

motor(left\_back) = drive\_power;

motor(left\_front) = drive\_power;

motor(right\_back) = drive\_power;

motor(right\_front) = drive\_power;

ClearTimer(T1);

ClearTimer(T2);

nMotorEncoder[hopper] = 0;

while((time100[T1] < (ScoreTime\*10)))

{

if (abs(EncoderReading - nMotorEncoder[hopper]) < 10 )

{

StartCount = 1;

}

else

{

StartCount = 0;

ClearTimer(T2);

}

if ((time100[T2] > 10) && (StartCount != 0) )

{

motor(hopper) = 15;

motor(left\_back) = 0;

motor(left\_front) = 0;

motor(right\_back) = 0;

motor(right\_front) = 0;

while((time100[T1] < (ScoreTime\*10)))

{

brush\_encoder = nMotorEncoder[brush];

wait10Msec(25);

if (abs(brush\_encoder - nMotorEncoder[brush]) < 10)

{

motor(hopper) = -40;

wait10Msec(25);

motor(hopper) = 20;

wait10Msec(5);

}

if (wiggle == 1)

{

for(int c =0 ; c < 10; c++)

{

motor(left\_back) = 15;

motor(left\_front) = 15;

motor(right\_back) = -15;

motor(right\_front) = -15;

wait10Msec(10);

}

for(int c =0 ; c < 10; c++)

{

motor(left\_back) = -15;

motor(left\_front) = -15;

motor(right\_back) = 15;

motor(right\_front) = 15;

wait10Msec(10);

}

for(int c =0 ; c < 10; c++)

{

motor(left\_back) = 8;

motor(left\_front) = 8;

motor(right\_back) = 8;

motor(right\_front) = 8;

wait10Msec(5);

}

}

}

}

// wait10Msec(10);

EncoderReading = nMotorEncoder[hopper];

if((abs(nMotorEncoder[hopper]) > MaxAngle))

{

motor(hopper) = -40;

wait10Msec(50);

nMotorEncoder[right\_front] = 0;

nMotorEncoder[left\_front] = 0;

nMotorEncoder[right\_back] = 0;

nMotorEncoder[left\_back] = 0;

wait10Msec(10);

motor[left\_front] = 30;

motor[right\_front] = 30;

motor[left\_back] = 30;

motor[right\_back] = 30;

wait10Msec(10);

while( ( abs(nMotorEncoder[left\_front]) < CountDistance ) &&

( abs(nMotorEncoder[right\_front]) < CountDistance ) )

{

}

motor[left\_front] = 0;

motor[right\_front] = 0;

motor[left\_back] = 0;

motor[right\_back] = 0;

wait10Msec(10);

motor(hopper) = 60;

motor(left\_back) = drive\_power;

motor(left\_front) = drive\_power;

motor(right\_back) = drive\_power;

motor(right\_front) = drive\_power;

}

}

motor(brush)= 0;

motor(hopper) = 0;

motor(left\_back) = 0;

motor(left\_front) = 0;

motor(right\_back) = 0;

motor(right\_front) = 0;

}

**Autonomous Programs:**

We have eight different programs for the autonomous part of the match. Five programs are ones that we call “different,” and three are ones we call “same.” The “different” ones (or diff in the program) are for the two starting squares where your alliance color does not match the nearest bridge. The “same” programs are for starting spots where your alliance color and the color of the nearest bridge match.

**DIFF\_B\_NP**

This program goes over the bridge and dispenses the high and middle dispensers.

task main()

{

float gyro\_offset = 0;

int inch;

int move\_err = 0;

nMaxRegulatedSpeed12V = 1000;

nMotorPIDSpeedCtrl[right\_front] = mtrNoReg;

nMotorPIDSpeedCtrl[right\_back] = mtrNoReg;

nMotorPIDSpeedCtrl[left\_front] = mtrNoReg;

nMotorPIDSpeedCtrl[left\_back] = mtrNoReg;

nMotorPIDSpeedCtrl[brush] = mtrNoReg;

motor(right\_front) = 1;

motor(right\_back) = 1;

motor(left\_front) = 1;

motor(left\_back) = 1;

motor(brush) = 1;

for( int c = 0; c < 20; c++)

{

gyro\_offset = gyro\_offset + SensorValue[Gyro];

wait10Msec(5);

}

gyro\_offset = gyro\_offset/20;

motor(right\_front) = 0;

motor(right\_back) = 0;

motor(left\_front) = 0;

motor(left\_back) = 0;

motor(brush) = 0;

wait10Msec(5);

nMotorPIDSpeedCtrl[right\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[right\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[brush] = mtrSpeedReg;

initializeRobot();

waitForStart();

//// Meat of Program /////////////////////////////////////

nMotorEncoder[hopper] = 0;

motor(hopper) = -50;

while (nMotorEncoder[hopper]> -200)

{

}

motor(hopper) = 0;

move(-18,40,gyro\_offset);

wait10Msec(5);

turn(-45, gyro\_offset);

wait10Msec(5);

move(-24,30, gyro\_offset);

wait10Msec(5);

move(10,40, gyro\_offset);

wait10Msec(5);

turn(135, gyro\_offset);

wait10Msec(5);

servo[Bridge\_L] = BL\_down;

servo[Bridge\_R] = BR\_down;

move(88, 70, gyro\_offset);

wait10Msec(5);

servo[Bridge\_L] = BL\_up;

servo[Bridge\_R] = BR\_up;

turn(-92, gyro\_offset);

wait10Msec(5);

inch = SensorValue[SonarLeft] / 2.54;

motor(hopper) = -50;

wait10Msec(150);

motor(hopper) = 0;

wait10Msec(5);

move(inch - 12, 30, gyro\_offset);

wait10Msec(5);

motor(right\_back)= 15;

motor(right\_front) = 15;

motor(left\_back) = 15;

motor(left\_front) = 15;

wait10Msec(50);

motor(right\_back)= 0;

motor(right\_front) = 0;

motor(left\_back) = 0;

motor(left\_front) = 0;

wait10Msec(5);

score(5, 0);

wait10Msec(5);

motor(brush) = 50;

move\_err = move(-53, 70, gyro\_offset);

wait10Msec(5);

motor(brush) = 0;

motor(hopper) = 50;

wait10Msec(75);

motor(hopper) = 0;

wait10Msec(5);

turn(90+move\_err, gyro\_offset);

wait10Msec(5);

IR\_align(gyro\_offset);

wait10Msec(5);

score(10, 1);

wait10Msec(5);

}

**DIFF\_B\_P**

This programs score preloads, goes over the bridge, and empties the middle dispenser.

task main()

{

float gyro\_offset = 0;

int move\_err = 0;

nMaxRegulatedSpeed12V = 1000;

nMotorPIDSpeedCtrl[right\_front] = mtrNoReg;

nMotorPIDSpeedCtrl[right\_back] = mtrNoReg;

nMotorPIDSpeedCtrl[left\_front] = mtrNoReg;

nMotorPIDSpeedCtrl[left\_back] = mtrNoReg;

nMotorPIDSpeedCtrl[brush] = mtrNoReg;

motor(right\_front) = 1;

motor(right\_back) = 1;

motor(left\_front) = 1;

motor(left\_back) = 1;

motor(brush) = 1;

for( int c = 0; c < 20; c++)

{

gyro\_offset = gyro\_offset + SensorValue[Gyro];

wait10Msec(5);

}

gyro\_offset = gyro\_offset/20;

motor(right\_front) = 0;

motor(right\_back) = 0;

motor(left\_front) = 0;

motor(left\_back) = 0;

motor(brush) = 0;

wait10Msec(5);

nMotorPIDSpeedCtrl[right\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[right\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[brush] = mtrSpeedReg;

initializeRobot();

waitForStart();

nMotorEncoder[hopper] = 0;

motor(hopper) = -50;

while (nMotorEncoder[hopper]> -200)

{

}

motor(hopper) = 0;

move(-18,40,gyro\_offset);

wait10Msec(5);

turn(-45, gyro\_offset);

wait10Msec(5);

move(-8,30, gyro\_offset);

wait10Msec(5);

servo[right\_tail] = 0;

motor[left\_front] = -15;

motor[right\_front] = -15;

motor[left\_back] = -15;

motor[right\_back] = -15;

wait10Msec(50);

servo[preload] = 75;

wait10Msec(100);

servo[preload] = 250;

servo[right\_tail] = 255;

wait10Msec(50);

move(6,40,gyro\_offset);

wait10Msec(5);

servo[Bridge\_L] = BL\_down;

servo[Bridge\_R] = BR\_down;

turn(135, gyro\_offset);

wait10Msec(5);

move(84, 70,gyro\_offset);

wait10Msec(5);

servo[Bridge\_L] = BL\_up;

servo[Bridge\_R] = BR\_up;

move(18, 70, gyro\_offset);

wait10Msec(5);

turn(-85, gyro\_offset);

wait10Msec(5);

move(-48, 70,gyro\_offset);

wait10Msec(5);

turn(85, gyro\_offset);

wait10Msec(5);

move(-18, 50, gyro\_offset);

wait10Msec(5);

IR\_align(gyro\_offset);

wait10Msec(5);

score(10, 1);

wait10Msec(5);

}

**DIFF\_M\_P**

This program scores preloads, goes over the mountain, and empties the middle dispenser.

task main()

{

float gyro\_offset = 0;

int move\_err = 0;

nMaxRegulatedSpeed12V = 1000;

nMotorPIDSpeedCtrl[right\_front] = mtrNoReg;

nMotorPIDSpeedCtrl[right\_back] = mtrNoReg;

nMotorPIDSpeedCtrl[left\_front] = mtrNoReg;

nMotorPIDSpeedCtrl[left\_back] = mtrNoReg;

nMotorPIDSpeedCtrl[brush] = mtrNoReg;

motor(right\_front) = 1;

motor(right\_back) = 1;

motor(left\_front) = 1;

motor(left\_back) = 1;

motor(brush) = 1;

for( int c = 0; c < 20; c++)

{

gyro\_offset = gyro\_offset + SensorValue[Gyro];

wait10Msec(5);

}

gyro\_offset = gyro\_offset/20;

motor(right\_front) = 0;

motor(right\_back) = 0;

motor(left\_front) = 0;

motor(left\_back) = 0;

motor(brush) = 0;

wait10Msec(5);

nMotorPIDSpeedCtrl[right\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[right\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[brush] = mtrSpeedReg;

initializeRobot();

waitForStart();

//// Meat of Program /////////////////////////////////////

nMotorEncoder[hopper] = 0;

motor(hopper) = -50;

while (nMotorEncoder[hopper]> -200)

{

}

motor(hopper) = 0;

move(-18,40, gyro\_offset);

wait10Msec(5);

turn(-45, gyro\_offset);

wait10Msec(5);

move(-8,30, gyro\_offset);

wait10Msec(5);

servo[right\_tail] = 0;

motor[left\_front] = -15;

motor[right\_front] = -15;

motor[left\_back] = -15;

motor[right\_back] = -15;

wait10Msec(50);

servo[preload] = 75;

wait10Msec(100);

servo[preload] = 250;

motor[left\_front] = 0;

motor[right\_front] = 0;

motor[left\_back] = 0;

motor[right\_back] = 0;

turn(45, gyro\_offset);

wait10Msec(5);

move(-24,40, gyro\_offset);

wait10Msec(5);

servo[right\_tail] = 255;

turn(90, gyro\_offset);

wait10Msec(5);

move\_err = move(84, 60, gyro\_offset);

wait10Msec(5);

turn(move\_err, gyro\_offset);

wait10Msec(5);

IR\_align(gyro\_offset);

wait10Msec(5);

score(15, 0);

wait10Msec(5);

}

**DIFF\_RG\_M**

This program gets the rolling goal to the other side of the field.

task main()

{

float gyro\_offset = 0;

int move\_err = 0;

nMaxRegulatedSpeed12V = 1000;

nMotorPIDSpeedCtrl[right\_front] = mtrNoReg;

nMotorPIDSpeedCtrl[right\_back] = mtrNoReg;

nMotorPIDSpeedCtrl[left\_front] = mtrNoReg;

nMotorPIDSpeedCtrl[left\_back] = mtrNoReg;

nMotorPIDSpeedCtrl[brush] = mtrNoReg;

motor(right\_front) = 1;

motor(right\_back) = 1;

motor(left\_front) = 1;

motor(left\_back) = 1;

motor(brush) = 1;

for( int c = 0; c < 20; c++)

{

gyro\_offset = gyro\_offset + SensorValue[Gyro];

wait10Msec(5);

}

gyro\_offset = gyro\_offset/20;

motor(right\_front) = 0;

motor(right\_back) = 0;

motor(left\_front) = 0;

motor(left\_back) = 0;

motor(brush) = 0;

wait10Msec(5);

nMotorPIDSpeedCtrl[right\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[right\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[brush] = mtrSpeedReg;

initializeRobot();

waitForStart();

move(-18,40, gyro\_offset);

wait10Msec(5);

turn(135, gyro\_offset);

wait10Msec(5);

servo[Bridge\_L] = BL\_down;

servo[Bridge\_R] = BR\_down;

move(10, 30, gyro\_offset);

wait10Msec(5);

arc\_turn(20, 40, 6, gyro\_offset);

wait10Msec(5);

arc\_turn(-60, 40, 10, gyro\_offset);

wait10Msec(5);

move(30, 100, gyro\_offset);

wait10Msec(50);

servo[Bridge\_L] = BL\_up;

servo[Bridge\_R] = BR\_up;

move(48, 100, gyro\_offset);

wait10Msec(5);

IR\_align(gyro\_offset);

wait10Msec(5);

score(15, 1);

wait10Msec(5);

}

**DIFF\_RG\_SG**

This program pushes the rolling goal into the stationary goal.

task main()

{

float gyro\_offset = 0;

int move\_err = 0;

nMaxRegulatedSpeed12V = 1000;

nMotorPIDSpeedCtrl[right\_front] = mtrNoReg;

nMotorPIDSpeedCtrl[right\_back] = mtrNoReg;

nMotorPIDSpeedCtrl[left\_front] = mtrNoReg;

nMotorPIDSpeedCtrl[left\_back] = mtrNoReg;

nMotorPIDSpeedCtrl[brush] = mtrNoReg;

motor(right\_front) = 1;

motor(right\_back) = 1;

motor(left\_front) = 1;

motor(left\_back) = 1;

motor(brush) = 1;

for( int c = 0; c < 20; c++)

{

gyro\_offset = gyro\_offset + SensorValue[Gyro];

wait10Msec(5);

}

gyro\_offset = gyro\_offset/20;

motor(right\_front) = 0;

motor(right\_back) = 0;

motor(left\_front) = 0;

motor(left\_back) = 0;

motor(brush) = 0;

wait10Msec(5);

nMotorPIDSpeedCtrl[right\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[right\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[brush] = mtrSpeedReg;

initializeRobot();

waitForStart();

move(-40, 60, gyro\_offset);

wait10Msec(5);

arc\_turn(90, -40, 6, gyro\_offset);

wait10Msec(5);

turn(90, gyro\_offset);

wait10Msec(5);

servo[Bridge\_L] = BL\_down;

servo[Bridge\_R] = BR\_down;

wait10Msec(5);

move(25, 30, gyro\_offset);

wait10Msec(5);

move(6, 15, gyro\_offset);

wait10Msec(5);

arc\_turn(60, 40, 12, gyro\_offset);

wait10Msec(5);

servo[Bridge\_L] = BL\_up;

servo[Bridge\_R] = BR\_up;

move(-33, 30, gyro\_offset);

wait10Msec(5);

turn(85, gyro\_offset);

wait10Msec(5);

move(48, 80, gyro\_offset);

wait10Msec(5);

turn(-60, gyro\_offset);

wait10Msec(5);

move(70, 70, gyro\_offset);

wait10Msec(5);

IR\_align(gyro\_offset);

wait10Msec(5);

score(15, 1);

wait10Msec(5);

}

**SAME\_B\_P**

This program scores preloads, goes over the bridge, and dispenses the middle dispenser.

task main()

{

float gyro\_offset = 0;

int inch;

int move\_err = 0;

nMaxRegulatedSpeed12V = 1000;

nMotorPIDSpeedCtrl[right\_front] = mtrNoReg;

nMotorPIDSpeedCtrl[right\_back] = mtrNoReg;

nMotorPIDSpeedCtrl[left\_front] = mtrNoReg;

nMotorPIDSpeedCtrl[left\_back] = mtrNoReg;

nMotorPIDSpeedCtrl[brush] = mtrNoReg;

motor(right\_front) = 1;

motor(right\_back) = 1;

motor(left\_front) = 1;

motor(left\_back) = 1;

motor(brush) = 1;

for( int c = 0; c < 20; c++)

{

gyro\_offset = gyro\_offset + SensorValue[Gyro];

wait10Msec(5);

}

gyro\_offset = gyro\_offset/20;

motor(right\_front) = 0;

motor(right\_back) = 0;

motor(left\_front) = 0;

motor(left\_back) = 0;

motor(brush) = 0;

wait10Msec(5);

nMotorPIDSpeedCtrl[right\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[right\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[brush] = mtrSpeedReg;

initializeRobot();

waitForStart();

//// Meat of Program /////////////////////////////////////

nMotorEncoder[hopper] = 0;

motor(hopper) = -50;

while (nMotorEncoder[hopper]> -200)

{

}

motor(hopper) = 0;

move(-27,40, gyro\_offset);

wait10Msec(5);

turn(45, gyro\_offset);

wait10Msec(5);

servo[left\_tail] = 0;

motor[left\_front] = -15;

motor[right\_front] = -15;

motor[left\_back] = -15;

motor[right\_back] = -15;

wait10Msec(250);

servo[preload] = 75;

wait10Msec(100);

servo[preload] = 250;

motor[left\_front] = 0;

motor[right\_front] = 0;

motor[left\_back] = 0;

motor[right\_back] = 0;

turn(-47, gyro\_offset);

wait10Msec(5);

servo[left\_tail] = 255;

move(5, 50, gyro\_offset);

wait10Msec(10);

servo[Bridge\_L] = BL\_down;

servo[Bridge\_R] = BR\_down;

turn(-90, gyro\_offset);

wait10Msec(5);

move\_err = move(80, 70, gyro\_offset);

wait10Msec(5);

servo[Bridge\_L] = BL\_up;

servo[Bridge\_R] = BR\_up;

turn(90 , gyro\_offset);

wait10Msec(5);

inch = SensorValue[SonarLeft] / 2.54;

motor(hopper) = -70;

wait10Msec(50);

move(inch - 12, 30, gyro\_offset);

wait10Msec(5);

motor(hopper) = 0;

wait10Msec(5);

motor(right\_back)= 15;

motor(right\_front) = 15;

motor(left\_back) = 15;

motor(left\_front) = 15;

wait10Msec(50);

motor(right\_back)= 0;

motor(right\_front) = 0;

motor(left\_back) = 0;

motor(left\_front) = 0;

wait10Msec(5);

score(5, 1);

wait10Msec(7);

motor(brush) = 75;

move(-53, 70, gyro\_offset);

wait10Msec(5);

motor(brush) = 0;

motor(hopper) = 80;

wait10Msec(100);

motor(hopper) = 0;

wait10Msec(5);

turn(-90, gyro\_offset);

wait10Msec(5);

IR\_align(gyro\_offset);

wait10Msec(5);

score(10, 1);

wait10Msec(5);

}

**SAME\_C\_NP**

This program goes over the cliff and dispenses the side and middle dispensers.

task main()

{

int gyro\_offset = 0;

int inch;

int move\_err=0;

nMaxRegulatedSpeed12V = 1000;

nMotorPIDSpeedCtrl[right\_front] = mtrNoReg;

nMotorPIDSpeedCtrl[right\_back] = mtrNoReg;

nMotorPIDSpeedCtrl[left\_front] = mtrNoReg;

nMotorPIDSpeedCtrl[left\_back] = mtrNoReg;

nMotorPIDSpeedCtrl[brush] = mtrNoReg;

motor(right\_front) = 1;

motor(right\_back) = 1;

motor(left\_front) = 1;

motor(left\_back) = 1;

motor(brush) = 1;

for( int c = 0; c < 20; c++)

{

gyro\_offset = gyro\_offset + SensorValue[Gyro];

wait10Msec(5);

}

gyro\_offset = gyro\_offset/20;

motor(right\_front) = 0;

motor(right\_back) = 0;

motor(left\_front) = 0;

motor(left\_back) = 0;

motor(brush) = 0;

wait10Msec(5);

nMotorPIDSpeedCtrl[right\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[right\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[brush] = mtrSpeedReg;

initializeRobot();

waitForStart();

///// Meat of the Program /////////////////////////////////////////////////////////

move(-3,40, gyro\_offset);

wait10Msec(5);

turn(-90, gyro\_offset);

wait10Msec(5);

move\_err = move(96, 70, gyro\_offset);

wait10Msec(5);

turn((90+move\_err), gyro\_offset);

wait10Msec(5);

move(-8,40, gyro\_offset);

wait10Msec(5);

inch = SensorValue[SonarLeft] / 2.54;

motor(hopper) = -50;

wait10Msec(150);

motor(hopper) = 0;

wait10Msec(5);

move(inch - 12, 30, gyro\_offset);

wait10Msec(5);

motor(right\_back)= 15;

motor(right\_front) = 15;

motor(left\_back) = 15;

motor(left\_front) = 15;

wait10Msec(50);

motor(right\_back)= 0;

motor(right\_front) = 0;

motor(left\_back) = 0;

motor(left\_front) = 0;

wait10Msec(5);

score(5, 0);

wait10Msec(5);

motor(brush) = 50;

move\_err = move(-53, 70, gyro\_offset);

wait10Msec(5);

motor(brush) = 0;

motor(hopper) = 50;

wait10Msec(75);

motor(hopper) = 0;

wait10Msec(5);

turn(-90+move\_err, gyro\_offset);

wait10Msec(5);

IR\_align(gyro\_offset);

wait10Msec(5);

score(10, 1);

wait10Msec(5);

}

**SAME\_M\_P**

This program scores preloads, goes over the mountain, and dispenses the side and middle dispensers.

task main()

{

float gyro\_offset = 0;

int move\_err = 0;

nMaxRegulatedSpeed12V = 1000;

nMotorPIDSpeedCtrl[right\_front] = mtrNoReg;

nMotorPIDSpeedCtrl[right\_back] = mtrNoReg;

nMotorPIDSpeedCtrl[left\_front] = mtrNoReg;

nMotorPIDSpeedCtrl[left\_back] = mtrNoReg;

nMotorPIDSpeedCtrl[brush] = mtrNoReg;

motor(right\_front) = 1;

motor(right\_back) = 1;

motor(left\_front) = 1;

motor(left\_back) = 1;

motor(brush) = 1;

for( int c = 0; c < 20; c++)

{

gyro\_offset = gyro\_offset + SensorValue[Gyro];

wait10Msec(5);

}

gyro\_offset = gyro\_offset/20;

motor(right\_front) = 0;

motor(right\_back) = 0;

motor(left\_front) = 0;

motor(left\_back) = 0;

motor(brush) = 0;

wait10Msec(5);

nMotorPIDSpeedCtrl[right\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[right\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_front] = mtrSpeedReg;

nMotorPIDSpeedCtrl[left\_back] = mtrSpeedReg;

nMotorPIDSpeedCtrl[brush] = mtrSpeedReg;

initializeRobot();

waitForStart();

//// Meat of Program /////////////////////////////////////

nMotorEncoder[hopper] = 0;

motor(hopper) = -50;

while (nMotorEncoder[hopper]> -200)

{

}

motor(hopper) = 0;

move(-27,40, gyro\_offset);

wait10Msec(5);

turn(45, gyro\_offset);

wait10Msec(5);

servo[left\_tail] = 0;

motor[left\_front] = -15;

motor[right\_front] = -15;

motor[left\_back] = -15;

motor[right\_back] = -15;

wait10Msec(250);

servo[preload] = 75;

wait10Msec(100);

servo[preload] = 250;

motor[left\_front] = 0;

motor[right\_front] = 0;

motor[left\_back] = 0;

motor[right\_back] = 0;

turn(-47, gyro\_offset);

wait10Msec(5);

move(-18, 50, gyro\_offset);

wait10Msec(10);

servo[left\_tail] = 255;

turn(-90, gyro\_offset);

wait10Msec(5);

move(78, 60, gyro\_offset);

wait10Msec(5);

IR\_align(gyro\_offset);

wait10Msec(5);

score(10, 1);

wait10Msec(5);

}

**Outreach:**

**Ocotillo Elementary School Presentation**

**June 4, 2010**

The purpose of this event was to introduce the entire 6th grade class at Ocotillo Elementary School to FIRST and robotics in a fun way (as part of Science Week at the school.) For 45 minutes, we drove our robot around and talked to the kids about FIRST, our team, and how to start your own team. The kids had so much fun!

**Highland High School Career Day**

**April 24, 2010**

We had a table at the school’s career day and talked to high-schoolers that attended; the rest of the event included other speakers and presentations. The students seemed interested, and I think this was a successful outreach event.

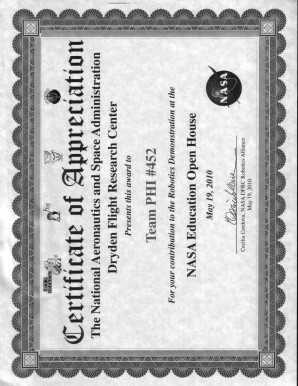
**NASA Education Open House/AERO Institute Grand Reopening**

**May 19, 2010**

This was a grand reopening event for the AERO Institute, a local center for space and aerospace education. The event featured displays and demonstrations by two local FRC teams and our team. We brought our robot and some of the field elements so we could demonstrate our scoring strategy. AERO Institute officials and NASA officials enjoyed driving our robot and trying to score waffle balls, and local students who attended were excited to see the robot in action. This was quite a successful outreach event.

Talking to AERO Institute employees, an Air Force general driving our robot



NASA gave us this certificate for participating in the NASA Open House/AERO Institute demo. At this time, the members of PHI teams #4322 and #452 were part of one team, PHI #452.**2010 VEX Workshop**

**June 21-25, 2010**

This was a 5-day workshop for 40 middle-schoolers designed to introduce participants to robotics by building robots (using the VEX system) to accomplish simple tasks. The game and tournament used for the event were designed by students and modeled on FIRST games and tournaments.

Students were divided into ten randomized teams, and the week began with icebreakers, an introduction to FIRST, Gracious Professionalism, and the VEX kit. Daily activities included team challenges from NASA and local educators, fun activities such as a remote-controlled helicopter and observation dome to get kids excited about science, presentations by NASA and AERO Institute representatives and robotics team members, and robot building sessions. All these activities combined to form a fun and exciting environment to introduce kids to robotics.

**LEGO Roadshow**

**August 28, 2010**

We helped a local FLL team with their one-day workshop they hosted for about 40 middle-schoolers. We installed NXT-G software on the FLL team’s computers and guided groups of 2 or 3 students through the process of building simple Lego robots to accomplish tasks. The event also included presentations about FLL and robotics.

**Super Science Saturday**

**October 23, 2010**

This yearly event, held at Joe Walker Middle School, was focused on getting elementary school kids excited about math and science, and it was full of activities to do just that. PHI Robotics joined two other local FIRST teams to demonstrate our robots and add some excitement to the event. Since we used this event as a build day as well as an outreach event, kids got to see our robots in progress and how we work to overcome challenges. There were a few kids who were really excited about what we were doing, and we were able to talk with them about joining an FLL team or even starting their own. Overall, this was a very successful outreach event.

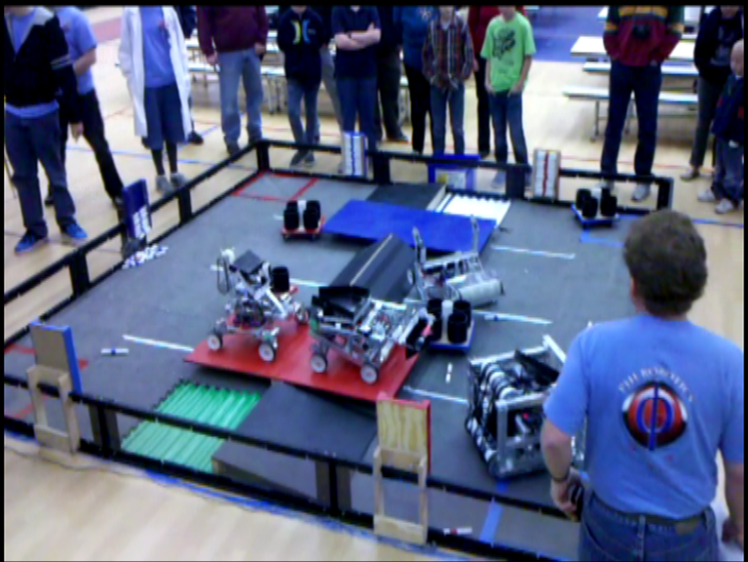
Talking to a mentor from another team, working on hardware and software at Super Science Saturday

**High Desert FTC Scrimmage**

**November 20, 2010**

Along with 2 other local FTC teams and the other PHI team (452), we organized an FTC scrimmage/build day where we ran matches, discussed our robots and how we could improve them, and made improvements to our autonomies and physical robots. This gave us practice with “real” matches (timed, with 3 other robots on the field) and allowed the other teams to practice with a full field.

Students from a local FLL team as well as reporters from the local newspaper attended, and a story appeared on the front page of the paper a few days later, so this event also helped introduce FTC to the community.

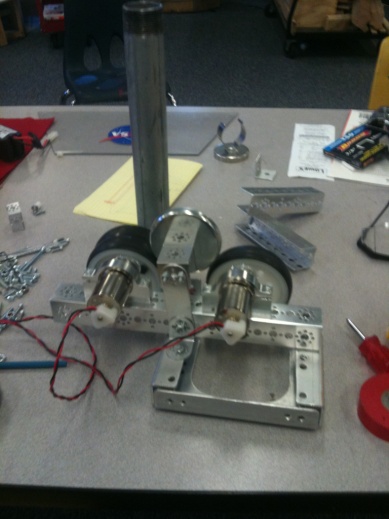
Before a match at the High Desert FTC Scrimmage

PHI Omega and PHI Alpha balance together at the end of a match



**Antelope Valley High School (FRC Team #2339)**

**January 21, 2011 and another day in January 2011**

We went to help a local FRC team get familiar with the Tetrix system that they needed to use to build their “mini-bot” for their FRC challenge. We also helped the Antelope Valley High School team prototype ideas for their “mini-bot.” We went back another day in January to provide further assistance with the Tetrix system and with the final design. The team we helped was FRC Team #2339 of Antelope Valley High School, Lancaster, California.



**Las Vegas Elementary School Outreach**

**January 28, 2011**

Along with three other FTC teams, we went to an elementary school in Las Vegas to demonstrate our robot, talk briefly about our team, and encourage the school’s FLL team. Another elementary school also attended via teleconference. The kids at the school assemblies watched this year’s FTC game video; then we discussed our team dynamic, the engineering process, and how we solved this year’s challenge. We also showed off our robots. After the assembly, we also visited the school’s FLL team to encourage them.

**Fundraising:**

**NASA:** Very generously paid for the robot kit and parts bought in previous years but continued to use this year. NASA also paid our $1000 World Championship registration fee.

**Northrop Grumman:** They gave us and the other PHI team (452) four computers. Paid for the travel for the 3 Northrop Grumman Mentors

**Sheriff Lee Baca:** Paid for metal parts we bought a few years ago, which we still continue to use.

**Jersey Mike’s Subs:** We did a fundraiser/demo at Jersey Mike’s Subs; we demonstrated our robot, and in exchange, they gave us 20% of their profit earned between 1:00 and 6:00 pm that day.

**Round Table Pizza:** This was similar to our Jersey Mike’s fundraiser, but in addition to demonstrating our robot, we also bussed tables at the restaurant in exchange for 10% of the profits.

**California Soft Tissue Therapy, Inc:** One of our team members’ dads owns this business, which donated to our team last year. We are still using the parts we bought with this money.

**Grace Chapel Ladies’ Tea:** We sold tins of homemade cookies and fudge at the Grace Chapel (local church) ladies’ tea.

**Garage Sale:** We hosted a garage sale with PHI Alpha and raised $751.28. This garage sale also helped a local family; in exchange for removing the items left over from their estate sale and giving them a tax receipt, we were allowed to sell the items at our garage sale. They really appreciated the helping hands, and we appreciated having stuff to sell at our garage sale!

**Personal Contributions:** Keith Inatomi, Mick Jaggers, James King, and Barry Strattan (Northrop Grumman employees) personally gave air miles, hotel points, and monetary donations. The air miles and hotel points allowed us to purchase a hotel room or flight to St. Louis for each team member.