Advanced FRC Programming

Kepler Sticka-Jones and Mark Van der Merwe

Introductions

- Kepler Sticka-Jones
  - Computer Science Major at the University of Utah
  - Graduated from Judge Memorial Catholic High School in Spring 2016
  - Former Lead Programmer for JudgeMent Call Robotics - Team 5933 (2016 Season Rookie Team and Utah FRC Finalist)
- Mark Van der Merwe
  - Computer Science Major at the University of Utah
  - Graduated from Academy for Math, Engineering, and Science in Spring 2016
  - Former Programming Team Lead for AMES Robotics - Team 3243

Agenda

1. Code Practices
    a. Command Based Programming vs. Individual Actions
        i. Creating comprehensive commands to use versus calling actions individually
    b. Sensors
        i. Using sensor feedback loops to make your bot smarter and easier to use
        ii. Autonomous functionality
2. Tooling
    a. Using Git and GitHub
    b. Using Gradle to Build
    c. Text Editors
    d. Demo

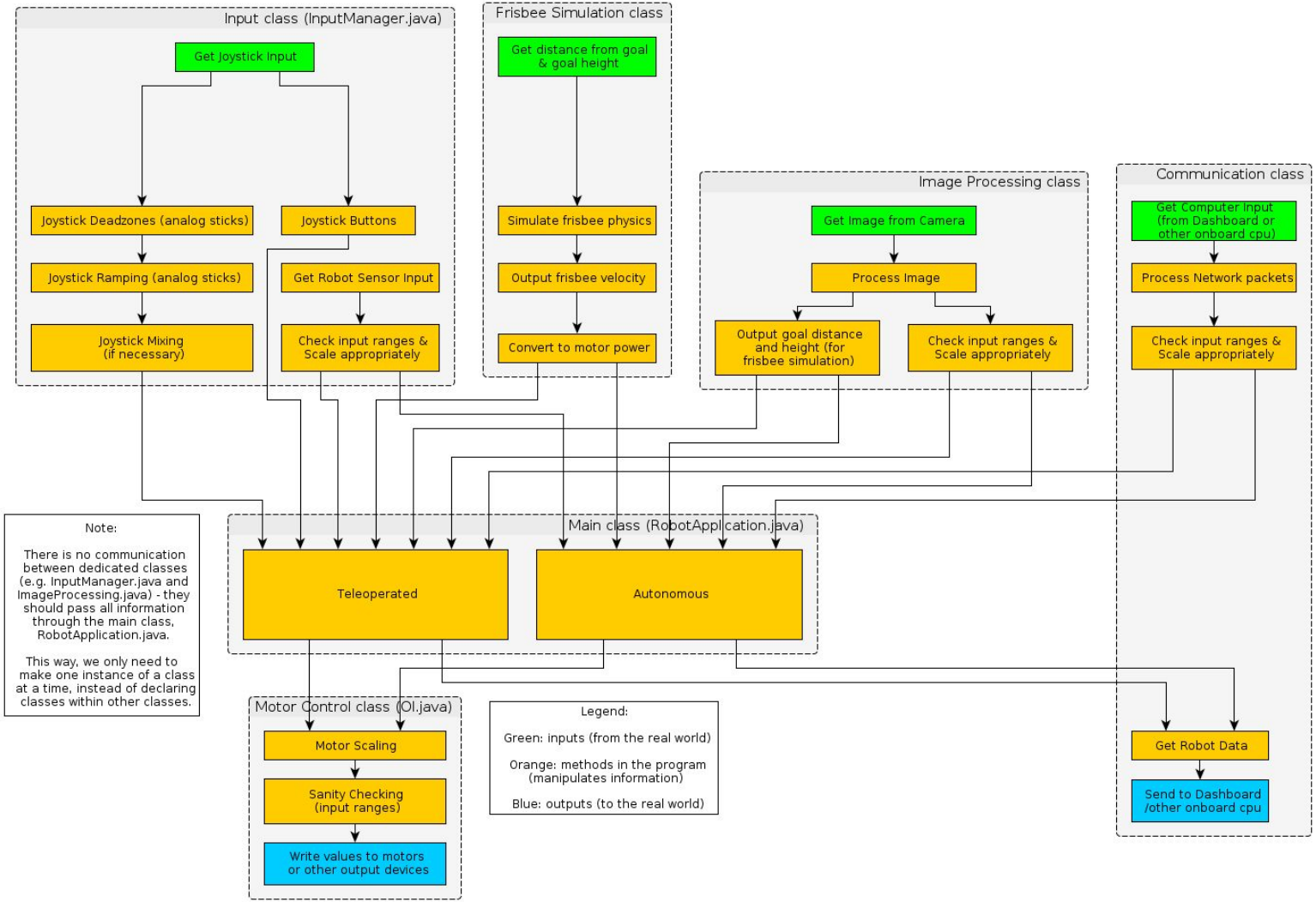
Code Practices

How should I organize my code?

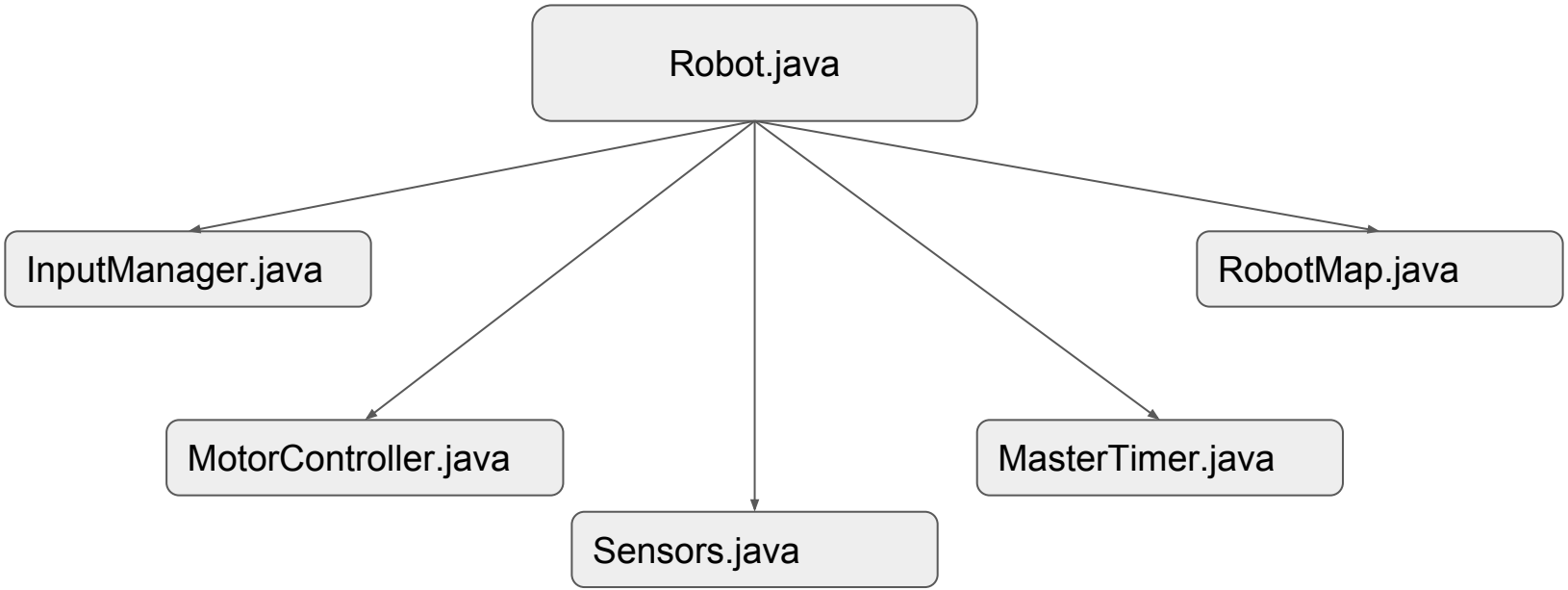
- Object-oriented is useful for many reasons, one being its ability to “modularize” code.
  - Classes, methods, variables, etc.
- For your robot, you can and *should* break your code into little bits.
- Three options for organizing code:
  - Iterative Robot - Robot code is set up with loops for the autonomous and teleop and you implement from there.
  - Command-Based Robot - Robot code is set up using subsystems and commands.
  - Sample Robot - This is mostly for testing where you need more control over program flow.

Iterative Robot

- Using this setup, you are left to decide how you want to manually separate and implement your code.
- You can choose how you want to modularize your code.


Input class (InputManager.java)
Get Joystick Input
Joystick Deadzones (analog sticks)
Joystick Buttons
Joystick Ramping (analog sticks)
Get Robot Sensor Input
Joystick Mixing (if necessary)
Check input ranges & Scale appropriately
Frisbee Simulation class
Get distance from goal & goal height
Simulate frisbee physics
Output frisbee velocity
Convert to motor power
Image Processing class
Get Image from Camera
Process Image
Output goal distance and height (for frisbee simulation)
Check input ranges & Scale appropriately
Communication class
Get Computer Input (from Dashboard or other onboard cpu)
Process Network packets
Check input ranges & Scale appropriately
Note:
There is no communication between dedicated classes (e.g. InputManager.java and ImageProcessing.java) - they should pass all information through the main class, RobotApplication.java.
This way, we only need to make one instance of a class at a time, instead of declaring classes within other classes.
Main class (RobotApplication.java)
Teleoperated
Autonomous
Motor Control class (OI.java)
Motor Scaling
Sanity Checking (input ranges)
Write values to motors or other output devices
Legend:
Green: inputs (from the real world)
Orange: methods in the program (manipulates information)
Blue: outputs (to the real world)
Get Robot Data
Send to Dashboard /other onboard cpu

Iterative Robot



Iterative Robot

- Within each class you have created, define the methods and variables each part requires.
- From main, you can call these methods from the classes.
- Loops repeat for the various sections of the game (teleop, autonomous).

```
    public void autonomousInit() {
        autoSelected = (String) chooser.getSelected();
//      autoSelected = SmartDashboard.getString("Auto Selector", defaultAuto);
        System.out.println("Auto selected: " + autoSelected);
    }

    /**
     * This function is called periodically during autonomous
     */
    public void autonomousPeriodic() {
        switch(autoSelected) {
        case customAuto:
        //Put custom auto code here
            break;
        case defaultAuto:
        default:
        //Put default auto code here
            break;
        }
    }

    /**
     * This function is called periodically during operator control
     */
    public void teleopPeriodic() {

    }

    /**
     * This function is called periodically during test mode
     */
    public void testPeriodic() {

    }

}
```

Iterative Robot Example

- AStarPathFinding.java
- AutoControl.java
- InputManager.java
- MotorControl.java
- Robot.java
- RobotMap.java
- Sensors.java

Robot.java

```
public void teleopPeriodic() {
    //DS.quickdrive(IM.input());
    DS.drive(IM.input());
    GB.pullIn(IM);
}
```

DriveSystem.java

```
void drive(double[] drv){
        cim1.set(-drv[0]);
        cim2.set(-drv[0]);//-drv[0]);
        cim3.set(drv[1]);//drv[1]);
        cim4.set(drv[1]);
}
```

InputManager.java

```
double[] input() {
        in[0] = ramp(deadZone(move.getRawAxis(1)));
        in[1] = ramp(deadZone(move.getRawAxis(3)));
        System.out.println(in[0]);
        System.out.println(in[1]);
        return in;
}
```

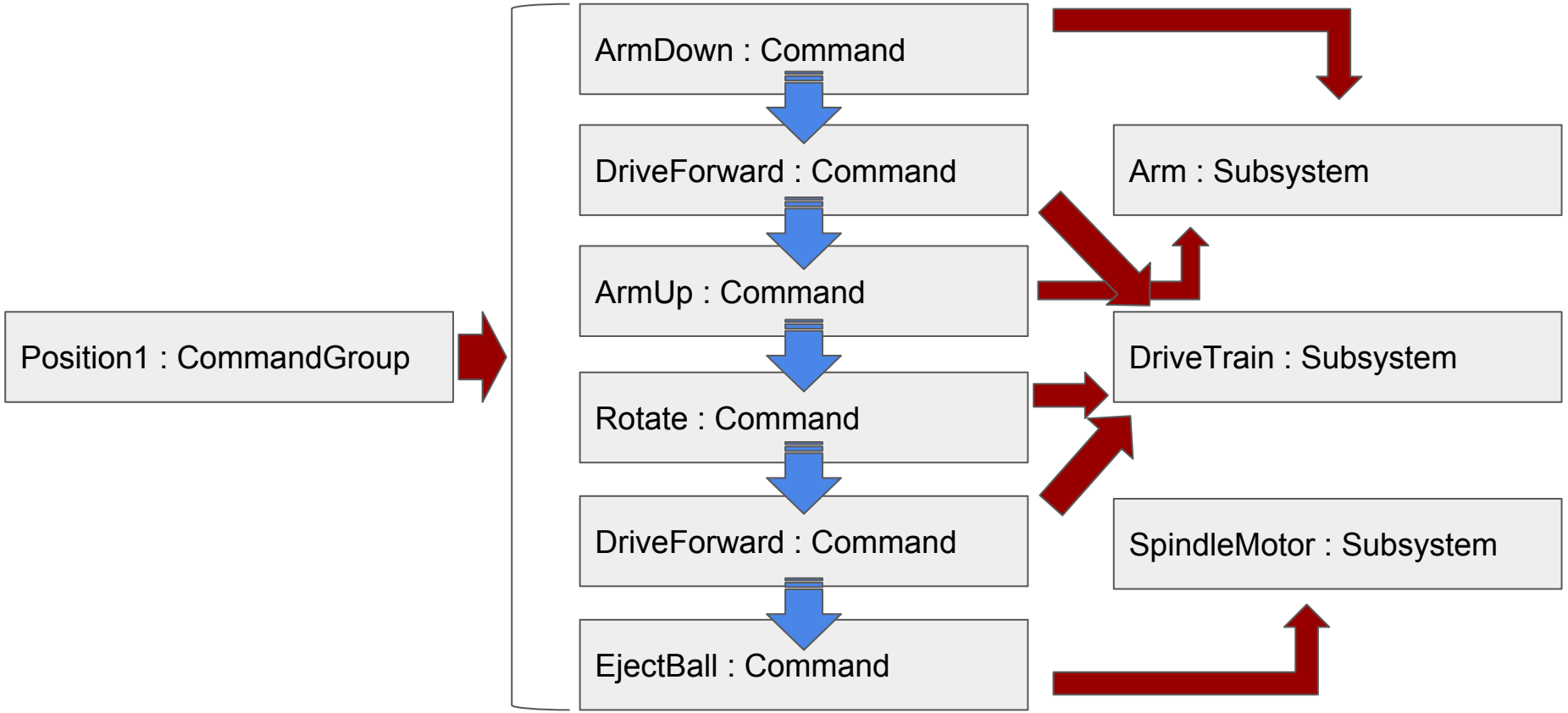
Command-Based Programming

- Divide your code into subsystems and commands.
- Subsystems - define the capabilities of each part of the robot and are subclasses of Subsystem.
- Commands - define the operation of the robot incorporating the capabilities defined in the subsystems. Commands are subclasses of Command or CommandGroup. Commands run when scheduled or in response to buttons being pressed or virtual buttons from the SmartDashboard.

Command-Based Programming

- Define each subsystem and command for your robot.
- Combine commands into command groups.
- Call these from the loops in main which again repeat for their specified portion of the match.

Command-Based Programming



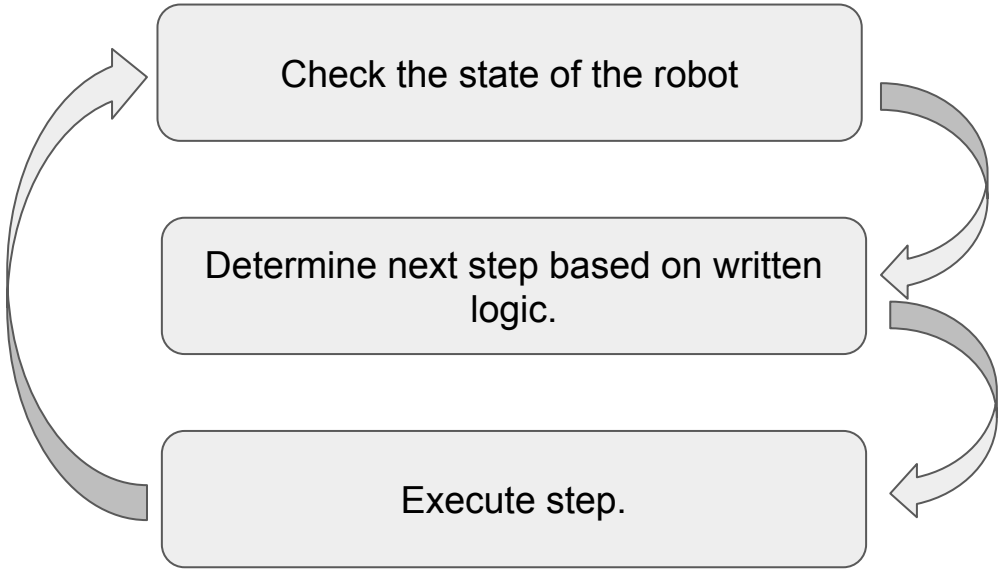
Command Example: Team 5933 Autonomous 2016

- Pre-programmed Path Command Classes
  - Built up with smaller commands
    - DriveStraight
      - Takes distance as a parameter
    - Rotate
      - Takes degrees as a parameter
    - EjectBall
      - Takes time as a parameter
    - RaiseArm/LowerArm
  - One for every possible starting position
- Autonomous command could be changed without recompiling
  - NetworkTable call and Switch statement in Robot initialization

Sensors

- FRC Supports a LOT of sensors:
  1. Gyroscope
  2. Potentiometer
  3. Accelerometer
  4. Encoders
  5. Switches
  6. Vision (cameras)
- How can you use sensors to make your robot as good as possible?

Sensor Feedback Loop



Sensor Feedback Loop

- Applications:
  - You already use it with your user input.
  - Develop autonomous entirely as a continuous sensor feedback loop.
  - Develop all your commands as something that at least in part is run by your sensors.
    - Shoot command aims based on a potentiometer that measure angle of launcher.
    - Move command determines when it has moved as far as it should based on encoder feedback.
- Allows driver to do less and your robot to do more.
  - Instead of the drivers having to try to aim, taking time and energy, the sensor feedback loops take care of it so all they have to do is drive and tell the program to shoot.
  - Instead of drivers having to pick up the ball manually, they can just let the sensors find the ball and pick it up for them.
- Implement into your current design manually, or try to use the PID Controller object, which can do some of the feedback loop automatically.

Sensor Loop Example: AMES Autonomous 2016

- Every time the autonomous loop was ran, we checked the state of our encoder and the gyro.
- Depending upon the two states, the program would decide what the next step should be.
  - Pre-programmed.
  - Algorithmically - Path-finding.
- Run the step.
- Repeat for the fifteen seconds.

Tooling

Git-flow + GitHub

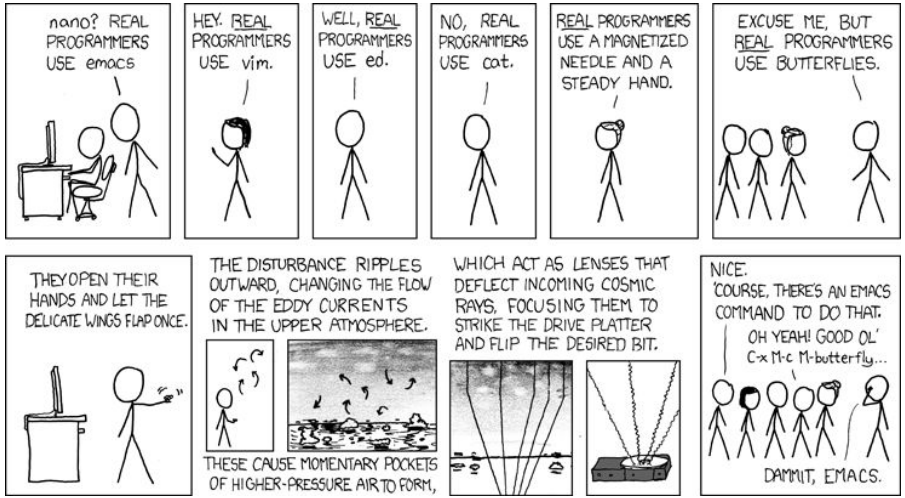
- Use Git
  - This gives you the opportunity to visualize how your code base grew and changed from build season to competition
  - Allows you to explain why code was changed without adding comments to code
  - Allows teammates to create changes independently of you
- Use GitHub
  - Host code for free
  - Share code with team mates
  - Share code with FIRST community
- Use GitHub Flow
  - Make branches for any group of changes
  - Make commits to the branch
  - When finished open a Pull Request on repo
  - Use Pull Request to discuss changes with team
  - Merge into master branch after discussion

Using Gradle to Build and Deploy

- Gradle
  - Used throughout the tech industry
    - The default build tool for Android applications
    - Used at Netflix, LinkedIn, and Twitter to build and deploy web services
    - Used to build countless open source projects
  - Manages dependencies and environment requirements
  - Manages deployments
  - Independent of text editor
  - Command-line and GUI driven
- GradleRIO
  - Plugin for Gradle and FIRST Robotics Competition projects
  - Maintained by team working closely with WPI, OpenRIO
  - Open source, and maintained on GitHub
  - Handles deployments to RoboRIO
  - Handles WPILib dependency, no need to install before

Text Editors

- Using Gradle allows you to code FRC projects outside of Eclipse
- IntelliJ IDEA has great customization ability and integrates neatly with Gradle
- Can use any other text editors, but does not guarantee Gradle integration or IDE features



Relevant xkcd