FIRST Team 3620
2013 Robot Redo Software Specification and Design

2013/09/29 20:53:13

Contents

1 Requirements

1.1 Autonomous Requirements

Operator can pick one of this from the SmartDashboard.

1.1.1 Box

Robot drives in a box.

1.1.2 Shoot3

Robot shoots 3 preloaded discs in place.
It will actuate the tilt, set the tilt to some angle (that comes from the dashboard), then in parallel starts the shooter wheels, indexes 1. After index is complete, double pump the flipper, index, double pump, index, double pump. Then turn shooter wheels off, and set the tilt to 0 degrees.

1.2 Teleop Requirements

1.2.1 Driving

The driver joystick will control the motion of the robot on the floor. The robot will usually just use 4 of the 6 motors; if the Turbo button is depressed, all 6 motors will be used. The driver joystick also has a "reverse" button that toggles the robot between forward and reverse mode; when in reverse, the robot drives as if the read of the robot is now the front.

1.2.2 Lift

There are 2 buttons on the driver's joystick. While the "up" button is pressed, the lift will extend, while the "down" button is pressed, the lift will retract.

1.2.3 Shooter

There is a button on the driver's joystick that when held, turns the shooter on. If the shooter is on, then the desired front speed is controller by driver's up and down buttons. There will be 2 speeds. The desired and actual RPM will be displayed on the dashboard.

1.2.4 Flipper

When the driver hits the shoot button, the flipper will move out for so long, wait for so long, retract for so long, then turn off. The control panel will have a manual override to work the flipper motorboth forwards and backwards.

1.2.5 Auger

If the switch at the bottom of the auger is hit, then the augur should index up one, unless there is a disk at the top. There will be a toggle switch on the control panel that will move the auger while the switch is actuated.

1.2.6 Harvester

There will be a button on the joystick that will toggle both the harvester wheels and the belts. The toggle switches on the control panel will override the current joystick button on/off setting. When the toggle is in the middle, the joystick on/off prevails.
Is there also a manual override for the harvester wheels?

1.2.7 ShooterTilt

There will be a button that turns the tilt motor on and off. If the tilt motor is on, then the desired angle is controlled by up and down buttons on the joystick. Each depression will move the tilt 30 degrees up or down. The limits of travel will be 0 and 60 degrees. The actual and desired tilt will be displayed on the dashboard.

2 Coding Guidelines

2.1 Capitalization

- Packages are all lower case: org.usfirst.frc3620.name (org.usfirst.frc3620.robotredo)
- Classes are capitalized and camel-cased.
- Methods are lower case and camel-cased.

2.2 Naming

- Projects are named FRC3620yyyyname (FRC36202013RobotRedo)
- Subsystem classes have Subsystem at the end of the name: DriveSubsystem.
- Command classes have Command at the end of the name, and the name of the subsystem they work with at the beginning DriveArcadeDriveCommand.
- If a command works several subsystems, then I don't know what to put at the beginning of it's name.
- Autonomous....

2.3 Command-Based Programming Common Practices

Generally, methods will be put into the subsystem to write to or read from the hardware; these routines should do as much as possible to disguise the hardware (in case we need to make a hardware change). If a Command is written so that it using an actual hardware device (DigitalIO, Victor, whatever), then something needs to be looked at.

- We will have a RobotMode class; it will be used to indicate what mode the robot is in (disabled, autonomous, teleop, test). This may be passed into `void periodic()` methods in the subsystems.
- There will be an `periodic(RobotMode robotmode)` method added to the Robot.java. The `teleopPeriodic()`, `autonomousPeriodic()`, `disabledPeriodic()`, and `testPeriodic()` should all call `periodic(RobotMode robotmode)`. This method will call the `periodic(RobotMode robotmode)` in each of the subsystems that need it.
- Each subsystem will have a `periodic(RobotMode robotmode)`, and `init()` method added to it. The subsystem's `periodic (RobotMode robotmode)` methods can update the dashboard, and possibly do other processing. `init()` will be used for one time processing, such as setting up custom Buttons that are actuated by on-robot hardware (custom Buttons that are actuated by operator interface will be set up in OI.java).

3 Design

3.1 Robot

3.2 Autonomous Commands

There will be two Autonomous commands: AutonomousBoxCommand and AutonomousShoot3Command.

3.3 DriveSubsystem

The default command will be the DriveArcadeCommand command.

3.3.1 Actuators, Controllers, and Sensors

PrimaryDrive is the drive for 4 of the 6 drive motors.
SecondaryDrive is the drive for the remaining 2 drive motors.
Gyro is the gyro used to measure turn.
DriveEncoder is the encoder used to measure distance traveled.

3.3.2 Commands

3.3.2.1 DriveArcadeCommand should be the default command for the subsystem.

requires	DriveSubsystem
initialization	None.
execute	Uses the arcadeDrive method to drive the robot.
isDone	Runs until interrupted.
end	Turns off drive.
interrupted	Same as end();

3.3.2.2 DriveTurnInPlaceCommand(double *degrees*) is used to turn *degrees* degrees. Negative values are to the left, positive values are to the right.

requires	DriveSubsystem
initialization	use resetGyro()
execute	Uses the arcadeDrive method to drive the robot.
isDone	finishes when the robot has turned far enough.
end	Turns off drive.
interrupted	Same as end();

3.3.2.3 DriveMoveInLineCommand(double *meters*) is used to move *meters* meters. Positive values are forward, negative values are backward.

requires	DriveSubsystem
initialization	use resetEncoder()
execute	Uses the arcadeDrive method to drive the robot.
isDone	finishes when the robot has moved far enough.
end	Turns off drive.
interrupted	Same as end();

3.3.2.4 DriveToggleReverseCommand() toggles "reverse mode".

requires	Nothing
initialization	None
execute	use the subsystem's "toggle reverse" method.
isDone	Runs once.

end Turns off drive.
interrupted Same as end();

3.3.3 Public Methods

`public void arcadeDrive (GenericHID hid, boolean turbo)`
drives the robot using just the PrimaryDrive if turbo is false, or both PrimaryDrive and SecondaryDrive if turbo is true.
`public void drive (double outputMagnitude, double curve)`
use the PrimaryDrive to.
`public void halt ()`
stop all drive motors.
`public void resetGyro ()`
reset the gyro.
`public double readGyro ()`
read the gyro heading in degrees.
`public void resetEncoder ()`
reset the encoder.
`public double readEncoder ()`
read the distance traveled in feet (meters, inches?)
`public void toggleReverseMode ()`
toggles the reverse mode flag.
`public void setReverseMode (boolean reverse)`
sets the reverse mode flag to *reverse*.
`public boolean getReverseMode()`
returns the current value of the reverse modem flag.

3.4 LiftSubsystem

3.4.1 Actuators, Controllers, and Sensors

ChinupController is the Victor controller for the chinup motors.

3.4.2 Commands

3.4.2.1 LiftExtendCommand extends the lift.
requires LiftSubsystem
initialization None.
execute Uses the liftExtend method to extend the lift.
isDone Runs until interrupted.
end liftHalt()
interrupted Same as end();

3.4.2.2 LiftRetractCommand retracts the lift.
requires LiftSubsystem
initialization None.
execute Uses the liftRetract method to retract the lift.
isDone Runs until interrupted.
end liftHalt()
interrupted Same as end();

3.4.3 Public Methods

`public void liftRetract()`
raises the robot (retracts the lift).
`public void liftExtend()`
lowers the robot (extends the lift).
`public void liftHalt()`
freezes the robot.

3.5 ShooterSubsystem

The front shooter runs at either 800 or 1600 RPM.

3.5.1 Actuators, Controllers, and Sensors

FrontShooterController is the Victor controller for the front shooter motor.
ReadShooterController is the Victor controller for the rear shooter motor.
FrontShooterCounter is the Counter for the front shooter motor.
FrontPID is the PID controller for the front shooter motor.

3.5.2 Commands

3.5.2.1 ShooterRunRearShooterCommand is the default command for the subsystem, it keeps the rear motor power set. The front motor power does not need to be set here because the PIDController keeps the front motor power set.

requires	ShooterSubSystem
initialization	None.
execute	Send the desired motor power (getDesiredRearMotorPower) to the rear shooter motor (setRearMotorPower())
isDone	Runs until interrupted.
end	shuts down the rear shooter motor (setRearMotorPower())
interrupted	Same as end();

3.5.2.2 ShooterFasterCommand bumps the desired RPM for the front shooter.

requires	nothing (we don't want to interrupt any commands)
initialization	None.
execute	Bumps the desired RPM up.
isDone	Runs once.
end	nothing
interrupted	Same as end();

3.5.2.3 ShooterSlowerCommand bumps the desired RPM for the front shooter.

requires	nothing (we don't want to interrupt any commands)
initialization	None.
execute	Bumps the desired RPM down.
isDone	Runs once.
end	nothing
interrupted	Same as end();

3.5.2.4 ShooterButtonCommand turns the shooters on while running.

requires	nothing (we don't want to interrupt any commands)
initialization	None.
execute	*needs work*

isDone Runs until interrupted.
end *needs work*
interrupted Same as end();

3.5.2.5 ShooterOnCommand turns the shooters on.

requires nothing (we don't want to interrupt any commands)
initialization None.
execute setRearShooterPower(), enableFrontPID()
isDone Runs once.
end nothing
interrupted Same as end();

3.5.2.6 ShooterOffCommand turns the shooters on.

requires nothing (we don't want to interrupt any commands)
initialization None.
execute setRearShooterPower(), disableFrontPID()
isDone Runs once.
end nothing
interrupted Same as end();

3.5.3 Public Methods

`public void setDesiredFrontRPM (double rpm)`
sets the desired speed for the front shooter motor.
`public double getDesiredFrontRPM ()`
gets the desired speed for the front shooter motor.
`public void setDesiredRearMotorPower (double power)`
sets the desired power for the rear shooter motor (saves the desired power).
`public double getDesiredRearMotorPower ()`
gets the desired power for the rear shooter motor.
`public double getCurrentFrontRPM ()`
gets the current speed for the front shooter motor.
`public enableFrontPID ()`
starts the PID controller for the front motor.
`public disableFrontPID ()`
stops the PID controller for the front motor.
`public setRearShooterPower(double power)`
set the power for rear shooter motor. Actually pokes the motor controller.
`public setFrontShooterPower(double power)`
set the power for front shooter motor. Actually pokes the motor controller. If the specified power is greater than 0.5, cap it at 0.5.

3.6 FlipperSubsystem

3.6.1 Actuators, Controllers, and Sensors

FlipperSpike is the Spike controller for the flipper motor.

3.6.2 Commands

3.6.2.1 FlipperFlipCommand extends the lift.

requires FlipperSubsystem

initialization save the current time.
execute Uses flipperForward() to move the flipper forward for so long, flipperHalt() for so long, flipperBackward() for so long.
isDone Runs until the flipperBackward at the end is done.
end flipperHalt()
interrupted Same as end();

3.6.2.2 FlipperAutonomousFlipCommand pumps the flipper multiple times.

requires FlipperSubsystem
initialization save the current time.
execute *needs work*
isDone Runs until the flipperBackward at the end is done.
end flipperHalt()
interrupted Same as end();

3.6.2.3 FlipperForwardCommand moves the flipper forward while running.

requires FlipperSubsystem
initialization None.
execute Uses flipperForward() to move the flipper forward.
isDone Runs until interrupted.
end flipperHalt()
interrupted Same as end();

3.6.2.4 FlipperBackwardCommand moves the flipper backwards while running.

requires FlipperSubsystem
initialization None.
execute Uses flipperBackward() to move the flipper backward.
isDone Runs until interrupted.
end flipperHalt()
interrupted Same as end();

3.6.3 Public Methods

`public void flipperForward()`
moves the flipper forward.
`public void flipperBackward()`
moves the flipper backward.
`public void flipperHalt()`
turns the flipper motor off.

3.7 AugerSubsystem

3.7.1 Actuators, Controllers, and Sensors

AugerController is the Victor controller for the chinup motors. *Chris: check the old robot; this could be a Spike. If so, make the change here.*
AugerLimitSwitch is the digital input for the limit switch.
AugerEncoder is the position encoder for the auger.

3.7.2 Commands

3.7.2.1 AugerIndexCommand indexes the auger one frisbee.

requires	AugerSubsystem
initialization	Record the current time.
execute	Uses the augerUp method to move the frisbees.
isDone	Runs when then auger is indexed and the elapsed time is more than 0.5 seconds.
end	Turns off drive (augerHalt())
interrupted	Same as end();

3.7.2.2 AugerUpCommand moves the auger in the up direction as long as it runs.

requires	AugerSubsystem
initialization	None.
execute	Uses the augerUp method to move the frisbees.
isDone	Runs until interrupted.
end	Turns off drive augerHalt())
interrupted	Same as end();

3.7.2.3 AugerDownCommand moves the auger in the down direction as long as it runs.

requires	AugerSubsystem
initialization	None.
execute	Uses the augerDown method to move the frisbees.
isDone	Runs until interrupted.
end	Turns off drive augerHalt())
interrupted	Same as end();

3.7.3 Public Methods

`public boolean readAugerLimitSwitch()`
reads the limit switch at the bottom of the auger.
`public boolean isAugerNeutral()`
tells if the auger is in the neutral position.
`public void augerUp()`
turn the auger motor so that frisbees move to the top.
`public void augerDown()`
turn the auger motor so that frisbees move to the bottom.
`public void augerHalt()`
turn the auger motor off.

3.7.4 Internal Methods

`double readAugerEncoder()`
reads the auger position in degrees. I'll bet that updateDashboard() and isAugerNeutral() will use this.

3.8 HarvesterSubsystem

3.8.1 Actuators, Controllers, and Sensors

HarvesterController is the Victor controller for the harvester motors.
BeltController is the Victor controller for the right harvester motor. *Chris: check the old robot; this could be a Spike. If so, make the change here.*

3.8.2 Commands

3.8.2.1 HarvesterRunCommand keeps the harvester and belt motors at the proper speed.

requires HarvesterSubsystem
initialization None.
execute *needs work*
isDone Runs until interrupted.
end *needs work*
interrupted Same as end();

3.8.2.2 HarvesterWheelsInManualCommand runs while the control panel HarvesterWheel override is set to move the wheels inward.

requires None.
initialization None.
execute *needs work*
isDone Runs until interrupted.
end *needs work*
interrupted Same as end();

3.8.2.3 HarvesterWheelsOutManualCommand runs while the control panel HarvesterWheel override is set to move the wheels outward.

requires None.
initialization None.
execute *needs work*
isDone Runs until interrupted.
end *needs work*
interrupted Same as end();

3.8.2.4 HarvesterBeltInManualCommand runs while the control panel Belt override is set to move the belt inward.

requires None.
initialization None.
execute *needs work*
isDone Runs until interrupted.
end *needs work*
interrupted Same as end();

3.8.2.5 HarvesterBeltOutManualCommand runs while the control panel Belt override is set to move the belt outward.

requires None.
initialization None.
execute *needs work*
isDone Runs until interrupted.
end *needs work*
interrupted Same as end();

3.8.2.6 HarvesterToggleCommand toggles whether the harvester is on or off.

requires None.
initialization None.
execute *needs work*
isDone Runs once.
end *needs work*

interrupted Same as end();

3.8.3 Public Methods

This needs work.
`public boolean setHarvesterPower(double power)`
set the power for the harvester motors.
`public boolean setBeltPower(double power)`
set the power for the belt motor.
`public void setOnOffState(boolean b)`
saves the current on/off state of the joystick button.
`public boolean getOnOffState()`
gets the current on/off state of the joystick button.
`public void setBeltDirection (Relay.Value direction)`
saves the desired belt direction. Relay.Value.kForward is inward, Relay.Value.kReverse is outward.
`public void setHarvesterDirection (Relay.Value direction)`
saves the desired harvester direction. Relay.Value.kForward is inward, Relay.Value.kReverse is outward.

3.9 ShooterTiltSubsystem

3.9.1 Actuators, Controllers, and Sensors

ShooterTiltController is the Victor controller for the shooter tilt motor.
ShooterTiltSensor is the analog input for the shooter tilt potentiometer.
ShooterTiltPID is the PID controller for the shooter tilt.

3.9.2 Commands

3.9.2.1 ShooterTiltBumpAngleUpCommand bumps the desired shooter angle up.

requires None.
initialization None.
execute *needs work*
isDone Runs once.
end *needs work*
interrupted Same as end();

3.9.2.2 ShooterTiltBumpAngleDownCommand bumps the desired shooter angle down.

requires None.
initialization None.
execute *needs work*
isDone Runs once.
end *needs work*
interrupted Same as end();

3.9.2.3 ShooterTiltEnableCommand enables the shooter tilt.

requires None.
initialization None.
execute *needs work*
isDone Runs once.
end *needs work*
interrupted Same as end();

3.9.2.4 ShooterTiltToggleCommand toggles the enabled/disabled state of the shooter tilt.

requires None.
initialization None.
execute *needs work*
isDone Runs once.
end *needs work*
interrupted Same as end();

3.9.2.5 ShooterTiltDisableCommand disables the shooter tilt.

requires None.
initialization None.
execute *needs work*
isDone Runs once.
end *needs work*
interrupted Same as end();

3.9.3 Public Methods

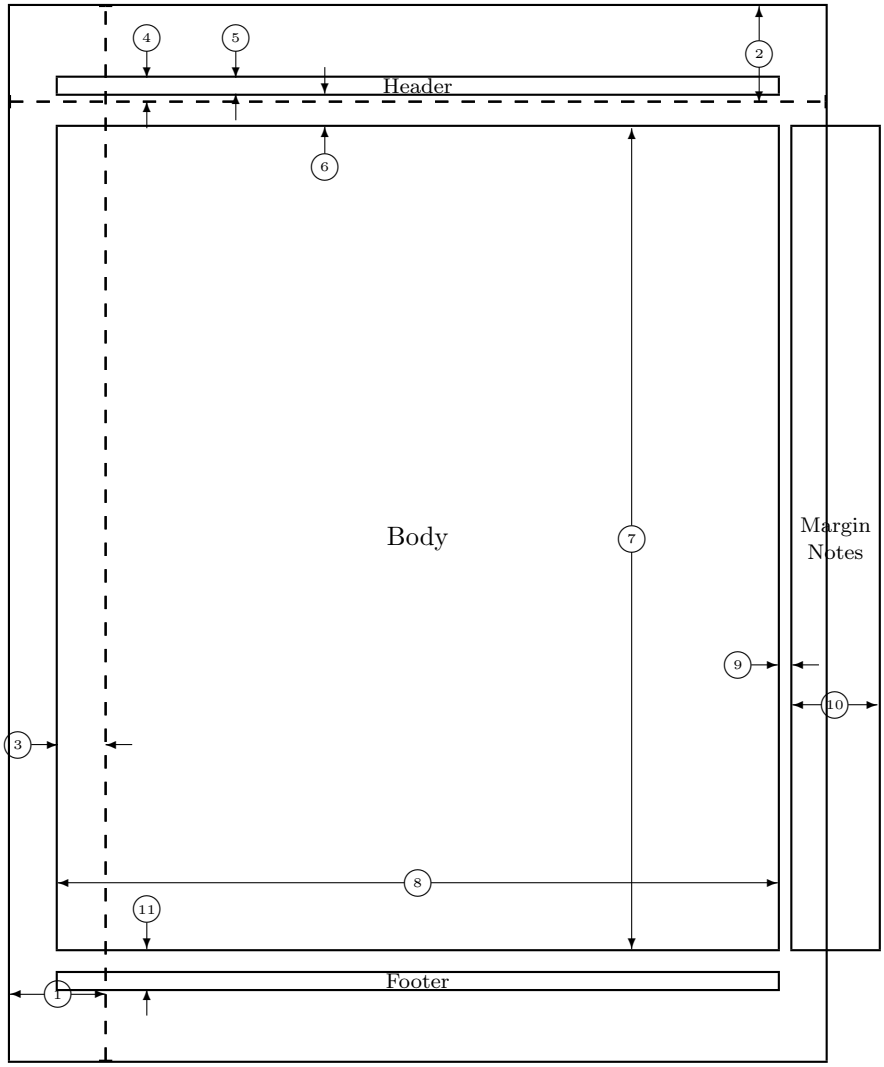
This needs work.
`public void enableShooterTilt()`
enables the shooter tilt.
`public void disableShooterTilt()`
disables the shooter tilt.
`public void toggleShooterTilt()`
toggles the shooter tilt.
`public void setDesiredShooterTiltAngle(double degrees)`
set (saves) the desired shooter tilt angle.
`public double getDesiredShooterTiltAngle()`
get the saved shooter tilt angle.
`public double getCurrentShooterTiltAngle()`
reads the current shooter tilt angle.

3.10 Operator Interface

Name	**Joystick**	Button Number	Command	When to Run
LiftExtend	Driver Joystick	?	LiftExtendCommand	whileHeld
LiftRetract	Driver Joystick	?	LiftRetractCommand	whileHeld
ShooterOn	Driver Joystick	?	ShooterOnCommand	whileHeld
ShooterFaster	Driver Joystick	?	ShooterFasterCommand	whenPressed
ShooterSlower	Driver Joystick	?	ShooterSlowerCommand	whenPressed
FlipperButton	Driver Joystick	?	FlipperFlipCommand	whenPressed
FlipperButton2	Control Panel	1	FlipperFlipCommand	whenPressed
FlipperForward	Control Panel	8	FlipperForwardCommand	whileHeld
FlipperBackward	Control Panel	7?	FlipperBackwardCommand	whileHeld
AugerUp	Control Panel	10	AugerUpCommand	whileHeld
AugerDown	Control Panel	9	AugerDownCommand	whileHeld
HarvesterOnOff	Driver Joystick	?	HarvesterToggleCommand	whenPressed
HarvesterIn	Control Panel	6	HarvesterWheelsInManualCommand	whileHeld
HarvesterOutt	Control Panel	5	HarvesterWheelsOutManualCommand	whileHeld
BeltIn	Control Panel	4	HarvesterBeltInManualCommand	whileHeld
BeltOut	Control Panel	3	HarvesterBeltOutManualCommand	whileHeld
ShooterTiltOnOff	Driver Joystick	?	ShooterTiltToggleCommand	whenPressed
ShooterTiltUp	Driver Joystick	?	ShooterTiltBumpAngleUpCommand	whenPressed
ShooterTiltDown	Driver Joystick	?	ShooterTiltBumpAngleDownCommand	whenPressed

A RobotMode.java

```
public class RobotMode {
   private String name;
   public RobotMode (String n) {
       name = n;
   }
   public String toString() {
       return name;
   }
   public static final RobotMode DISABLED = new RobotMode("DISABLED");
   public static final RobotMode AUTONOMOUS = new RobotMode("AUTONOMOUS");
   public static final RobotMode TELEOP = new RobotMode("TELEOP");
   public static final RobotMode TEST = new RobotMode("TEST");
}
```



1	one inch + \hoffset	2	one inch + \voffset
3	\oddsidemargin = -36pt	4	\topmargin = -18pt
5	\headheight = 12pt	6	\headsep = 25pt
7	\textheight = 619pt	8	\textwidth = 542pt
9	\marginparsep = 11pt	10	\marginparwidth = 65pt
11	\footskip = 30pt		\marginparpush = 5pt (not shown)
	\hoffset = 0pt		\voffset = 0pt
	\paperwidth = 614pt		\paperheight = 794pt