1 /\*

2 Waterloo Engineering Expeller of Dominoes Main Program 3

4 Sean Aitken, Henrique Engelke, Josh Morcombe, and Andor Siegers 5

6 v1.7

7

1. Assumptions:
2. - more than 3 instructions will be in instruction file
3. - no more than 100 instructions will be given to the robot
4. - robot is fully loaded at program start, with exactly 30 dominoes
5. in the hopper
6. - door is closed, dispenser arm is all the way back at program start
7. - if user is selecting line follow mode, it must be placed on a line of adequate length
8. with white on either side
9. - if user is selecting file follow mode, a file of the correct format must be
10. loaded on the robot 18
11. Motor Ports:
12. A - left drive wheel
13. B - dispenser motor
14. C - gate motor
15. D - right drive wheel 24
16. Sensor Ports:
17. 1 - MUX
18. 2 - gyro
19. 3 - touch
20. 4 - ultrasonic
21. \*/

31

1. #**include** "PC\_FileIO.c";
2. #**include** "mindsensors-ev3smux.h"
3. #**include** "UW\_sensorMux.c"

35

# typedef struct

1. {
2. **bool** is\_ang;
3. **int** val;

40

41 } Instr;

42

1. // one-time functions
2. **void configureAllSensors**(**bool** mode);
3. **bool selectMode**();
4. **void endProgram**();

47

1. // high level functions
2. **void followLine**(**bool** &drop\_index, **int** &domino\_count); // Sean
3. **void followPathFromFile**(**bool** &drop\_index, **int** &domino\_count); // Andor
4. **int getInstrFromFile**(Instr\* all\_instr);
5. **void dropDomino**(**bool** &drop\_index, **int** &domino\_count); // Henrique
6. **void somethingInTheWay**(**int** motor\_power); // stops and informs the user to move the object in the way
7. **void somethingInTheWay** (**int** left\_mot\_pow, **int** right\_mot\_pow); 55
8. // calculation functions
9. **int distToDeg**(**float** dist);
10. **float degToDist**(**int** deg);
11. **float average**(**int** value1, **int** value2); 60

61

1. // movement functions
2. **void setDriveTrainSpeed**(**int** speed);
3. **void driveDist**(**float** dist,**int** mot\_pow);
4. **void driveWhileDropping**(**float** dist, **int** mot\_pow, **bool** &drop\_index, **int** &domino\_count, **float**

&dist\_since\_last\_dom); // Andor

1. **void turnInPlace**(**int** angle, **int** mot\_pow);
2. **void turnWhileDropping**(**int** angle, **int** speed, **bool** &drop\_index, **int** &domino\_count, **float**

&dist\_since\_last\_dom); // Andor

|  |  |  |
| --- | --- | --- |
| 68 | **void** | **stopAndKnock**(); // Josh |
| 69 | **void** | **openDoor**(); |
| 70 | **void** | **closeDoor**(); |
| 71 |  |  |

1. // constants
2. **const float** WHEEL\_RAD = 2.75; // in cm
3. **const int** DOMINOS\_AT\_MAX\_LOAD = 30;
4. **const int** MAX\_INSTR = 100;
5. **const float** PIXELS\_PER\_CM = 5.0;
6. **const float** DIST\_BETWEEN\_DOMINOS = 3.75; // in cm
7. **const float** DIST\_BET\_DOM\_TURNING = 5.5; // in cm
8. **const int** DRIVE\_SPEED = 20; // for path from file mode
9. **const int** DIST\_IN\_FRONT\_LIM = 20; // in cm
10. **const float** TURN\_RAD = 20; // in cm - needs to be more than 6.75cm
11. **const int** TIME\_TO\_PRESS = 10; // in seconds
12. **const int** DOOR\_ANG = 90; // degrees
13. **const int** DOOR\_SPEED = 50;
14. **const int** DROP\_WAIT = 500; // in milliseconds
15. **const int** MUX\_WAIT = 10;
16. **const int** DISPENSER\_SPEED = -30;
17. **const int** DISPENSER\_POS0 = 80;
18. **const int** DISPENSER\_POS1 = -370;
19. **const int** DISPENSER\_POS2 = -530;
20. **const int** KNOCK\_SPEED = -15;

92

1. // port assignments
2. **const int** TOUCH\_PORT = S2;
3. **const int** GYRO\_PORT = S3;
4. **const int** MULTIPLEXER\_PORT = S1;
5. **const int** ULTRASONIC\_PORT = S4;

98

1. **const int** RIGHT\_MOT\_PORT = motorD;
2. **const int** LEFT\_MOT\_PORT = motorA;
3. **const int** DOOR\_MOT\_PORT = motorB;
4. **const int** DISPENSER\_MOT\_PORT = motorC; 103
5. task **main**()
6. {
7. // initialization for domino dropping
8. nMotorEncoder(DISPENSER\_MOT\_PORT) = 0;
9. nMotorEncoder(DOOR\_MOT\_PORT) = 0;
10. **bool** drop\_index = false; // false for back position, true for middle position
11. **int** domino\_count = DOMINOS\_AT\_MAX\_LOAD; 111

112 **if**(selectMode())// false for line follow, true for file path

113

114

115

116

117

118

119

120 }

121

{

}

# else

{

}

followPathFromFile(drop\_index, domino\_count);

followLine(drop\_index, domino\_count);

122 // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* one-time functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# void configureAllSensors(bool mode)

1. {
2. SensorType[TOUCH\_PORT] = sensorEV3\_Touch;
3. SensorType[GYRO\_PORT] = sensorEV3\_Gyro;
4. wait1Msec(50);
5. SensorType[ULTRASONIC\_PORT] = sensorEV3\_Ultrasonic;
6. wait1Msec(50);
7. SensorMode[GYRO\_PORT] = modeEV3Gyro\_Calibration;
8. wait1Msec(50);
9. SensorMode[GYRO\_PORT] = modeEV3Gyro\_RateAndAngle;
10. wait1Msec(50);

134

1. // if line follow mode is selected, configure sensors required for
2. // this mode
3. **if**(!mode)
4. {
5. SensorType[MULTIPLEXER\_PORT] = sensorEV3\_GenericI2C;
6. wait1Msec(100);

141

1. **if** (!initSensorMux(msensor\_S1\_1, colorMeasureColor))
2. {
3. displayString(2,"Failed to configure colour1");

# return;

146 }

1. wait1Msec(50);
2. **if** (!initSensorMux(msensor\_S1\_2, colorMeasureColor))
3. {
4. displayString(4,"Failed to configure colour2");

# return;

152 }

153 wait1Msec(50);

154 }

155 }

156

# bool selectMode()

1. {
2. displayTextLine(5, "Choose Mode");
3. displayTextLine(7, "Left - Follow Line");
4. displayTextLine(9, "Right - Follow Path from File"); 162
5. **while**(!getButtonPress(buttonLeft) && !getButtonPress(buttonRight))
6. {}

165

1. // returns true if buttonRight is pressed (path from file mode)
2. // returns false if buttonLeft is pressed (line follow mode)
3. **bool** mode = getButtonPress(buttonRight);
4. configureAllSensors(mode);

|  |  |  |  |
| --- | --- | --- | --- |
| 170 | wait1Msec(700); | |  |
| 171 | **return** mode; | |
| 172 | } | |
| 173 |  | |
| 174 | **void endProgram**() | |
| 175 | { | |
| 176 | setDriveTrainSpeed(0); | |
| 177 | time1[T1] = 0; | |
| 178 | // wait for user to press touch sensor | |
| 179 | **while**(time1[T1] < TIME\_TO\_PRESS\*1000) | |
| 180 | { | |
| 181 | **if**(SensorValue[TOUCH\_PORT]) | |
| 182 | stopAndKnock(); | |
| 183 | } | |
| 184 | stopAllTasks(); | |
| 185 | } | |
| 186 |  | |
| 187 | // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* high level functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |
| 188 | **void followLine**(**bool** &drop\_index, **int** &domino\_count) // Sean | |
| 189 | { | |
| 190 | time1[T2] = 0; | |
| 191 | **int** index = 0; | |
| 192 | **int** index2 = 0; | |
| 193 | **int** sensor1 = 0; | |
| 194 | **int** sensor2 = 0; | |
| 195 | **int** domino\_encoder\_spacing = distToDeg(DIST\_BETWEEN\_DOMINOS); | |
| 196 |  | |
| 197 | openDoor(); | |
| 198 |  | |
| 199 | **while**((domino\_count>0)&&(SensorValue(TOUCH\_PORT) == 0)) | |
| 200 | { | |
| 201 |  | |
| 202 | **if**((SensorValue[ULTRASONIC\_PORT]) < (DIST\_IN\_FRONT\_LIM)) | |
| 203 | { | |
| 204 | somethingInTheWay(0); | |
| 205 | } | |
| 206 |  | |
| 207 | **if**((average(nMotorEncoder[RIGHT\_MOT\_PORT],nMotorEncoder[LEFT\_MOT\_PORT])) domino\_encoder\_spacing) | | > |
| 208 | { |  | |
| 209 |  | dropDomino(drop\_index, domino\_count); | |
| 210 |  | nMotorEncoder[RIGHT\_MOT\_PORT] = nMotorEncoder[LEFT\_MOT\_PORT] = 0; | |
| 211 | } | | |
| 212 |  | | |
| 213 | motor[LEFT\_MOT\_PORT] = motor[RIGHT\_MOT\_PORT] = -10; | | |
| 214 |  | | |
| 215 | **if**(time1[T2] > index) | | |
| 216 | { | | |
| 217 | sensor1 = readMuxSensor(msensor\_S1\_1); | | |
| 218 | index = time1[T2] + MUX\_WAIT; | | |
| 219 |  | | |
| 220 | **if**(sensor1 == (**int**) colorBlack) | | |
| 221 | { | | |
| 222 | motor[RIGHT\_MOT\_PORT] = 0; | | |
| 223 | } | | |
| 224 | } | | |
| 225 |  | | |
| 226 | **if**(time1[T2] > index2) | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 227 |  | { |  |
| 228 |  | sensor2 = readMuxSensor(msensor\_S1\_2); |
| 229 |  | index2 = time1[T2] + MUX\_WAIT + 5; |
| 230 |  |  |
| 231 |  | **if**(sensor2 == (**int**) colorBlack) |
| 232 |  | { |
| 233 |  | motor[LEFT\_MOT\_PORT] = 0; |
| 234 |  | } |
| 235 |  | } |
| 236 |  | } |
| 237 |  | **if**(SensorValue(TOUCH\_PORT)) |
| 238 |  | { |
| 239 |  | stopAndKnock(); |
| 240 |  | } |
| 241 |  | endProgram(); |
| 242 | } |  |
| 243 |  |  |
| 244 | **void** | **followPathFromFile**(**bool** &drop\_index, **int** &domino\_count) // Andor |
| 245 | { |  |
| 246 |  | Instr all\_instr[MAX\_INSTR]; |
| 247 |  | **float** dist\_since\_last\_dom = 0; |
| 248 |  |  |
| 249 |  | **int** num\_instr = getInstrFromFile(all\_instr); |
| 250 |  |  |
| 251 |  | **int** num\_turns = 0; |
| 252 |  | **int** instr\_index = 0; |
| 253 |  |  |
| 254 |  | // drive to starting position |
| 255 |  | **while**(num\_turns < 2) |
| 256 |  | { |
| 257 |  | **if**(all\_instr[instr\_index].is\_ang) |
| 258 |  | { |
| 259 |  | num\_turns++; |
| 260 |  | turnInPlace(all\_instr[instr\_index].val, 20); |
| 261 |  | } |
| 262 |  | **else** |
| 263 |  | { |
| 264 |  | driveDist(all\_instr[instr\_index].val/PIXELS\_PER\_CM, | 50); |
| 265 |  | } |  |
| 266 |  | instr\_index++; |  |
| 267 |  | } |  |
| 268 |  |  |  |
| 269 | **while**(instr\_index < num\_instr && domino\_count > 0) | | |
| 270 | { | | |
| 271 | // loop through all instructions | | |
| 272 |  | | |
| 273 | **if**(all\_instr[instr\_index].is\_ang) | | |
| 274 | { | | |
| 275 | // turn | | |
| 276 | turnWhileDropping(all\_instr[instr\_index].val, DRIVE\_SPEED, drop\_index, domino\_count, dist\_since\_last\_dom); | | |
| 277 | } | | |
| 278 | **else** | | |
| 279 | { | | |
| 280 | // drive length | | |
| 281 | driveWhileDropping(all\_instr[instr\_index].val/PIXELS\_PER\_CM, DRIVE\_SPEED, drop\_index, domino\_count, dist\_since\_last\_dom); | | |
| 282 | } | | |

283 instr\_index++;

284 }

285

286 }

287

endProgram();

1. **int getInstrFromFile**(Instr\* all\_instr) // Andor
2. {
3. // open file and initialize variables
4. TFileHandle fin;
5. **bool** fileOkay = openReadPC(fin,"instr.txt"); 293
6. **int** num\_instr = 0;
7. readIntPC(fin, num\_instr);

296

1. **int** temp\_is\_ang\_int = 0;
2. **bool** temp\_is\_ang = false;
3. **int** temp\_val = 0;

300

1. **for**(**int** read\_index = 0; read\_index < num\_instr; read\_index++)
2. {
3. // read in instruction
4. readIntPC(fin, temp\_is\_ang\_int);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 305 | **if**(temp\_is\_ang\_int == | | | 0) |
| 306 | { | | |  |
| 307 | temp\_is\_ang = | | | false; |
| 308 | } | | |  |
| 309 | **else** | | |  |
| 310 | { | | |  |
| 311 | temp\_is\_ang = | | | true; |
| 312 | } | | |  |
| 313 |  | | |  |
| 314 |  | | readIntPC(fin, temp\_val); | |
| 315 |  | | all\_instr[read\_index].is\_ang = temp\_is\_ang; | |
| 316 |  | | all\_instr[read\_index].val = temp\_val; | |
| 317 | } | |  | |
| 318 |  | |  | |
| 319 |  | closeFilePC(fin); | | |
| 320 |  | **return** num\_instr; | | |
| 321 | } |  | | |
| 322 |  |  | | |
| 323 | **void** | **dropDomino**(**bool** &drop\_index, **int** &domino\_count) // Henrique | | |
| 324 | { |  | | |
| 325 |  | setDriveTrainSpeed(0); | | |
| 326 |  | closeDoor(); | | |
| 327 |  |  | | |
| 328 |  | // moves dispenser arm to next position, depending on current | | |
| 329 |  | // position | | |
| 330 |  | **if** (!drop\_index) | | |
| 331 |  | { | | |
| 332 |  | motor[DISPENSER\_MOT\_PORT] = DISPENSER\_SPEED; | | |
| 333 |  | **while** (nMotorEncoder(DISPENSER\_MOT\_PORT) > DISPENSER\_POS1) | | |
| 334 |  | { | | |
| 335 |  | // scan for touch press | | |
| 336 |  | **if**(SensorValue[TOUCH\_PORT]) | | |
| 337 |  | { | | |
| 338 |  | motor[DISPENSER\_MOT\_PORT] = 0; | | |
| 339 |  | stopAndKnock(); | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 340 |  | } |  |
| 341 |  | } |
| 342 |  | motor[DISPENSER\_MOT\_PORT] = 0; |
| 343 |  | drop\_index = true; |
| 344 |  | wait1Msec(DROP\_WAIT); |
| 345 |  | } |
| 346 |  | **else** |
| 347 |  | { |
| 348 |  | motor[DISPENSER\_MOT\_PORT] = DISPENSER\_SPEED; |
| 349 |  | **while** (nMotorEncoder(DISPENSER\_MOT\_PORT) > DISPENSER\_POS2) |
| 350 |  | { |
| 351 |  | **if**(SensorValue[TOUCH\_PORT]) |
| 352 |  | { |
| 353 |  | motor[DISPENSER\_MOT\_PORT] = 0; |
| 354 |  | stopAndKnock(); |
| 355 |  | } |
| 356 |  | } |
| 357 |  | motor[DISPENSER\_MOT\_PORT]= 0; |
| 358 |  |  |
| 359 |  | drop\_index = false; |
| 360 |  | wait1Msec(100); |
| 361 |  |  |
| 362 |  | // reset arm to initial position |
| 363 |  | motor[DISPENSER\_MOT\_PORT] = -DISPENSER\_SPEED; |
| 364 |  | **while** (nMotorEncoder(DISPENSER\_MOT\_PORT) < DISPENSER\_POS0) |
| 365 |  | { |
| 366 |  | **if**(SensorValue[TOUCH\_PORT]) |
| 367 |  | { |
| 368 |  | motor[DISPENSER\_MOT\_PORT] = 0; |
| 369 |  | stopAndKnock(); |
| 370 |  | } |
| 371 |  | } |
| 372 |  | motor[DISPENSER\_MOT\_PORT] = 0; |
| 373 |  | } |
| 374 |  | openDoor(); |
| 375 |  | domino\_count--; |
| 376 | } |  |
| 377 |  |  |
| 378 | **void** | **somethingInTheWay** (**int** motor\_power) // Josh |
| 379 | { |  |
| 380 |  | // Stops motors, displays message and plays a sound. Exits when object | is moved. |
| 381 |  | **while**(SensorValue[ULTRASONIC\_PORT] < DIST\_IN\_FRONT\_LIM) |  |
| 382 |  | { |  |
| 383 |  | setDriveTrainSpeed(0); |  |
| 384 |  | eraseDisplay(); |  |
| 385 |  | displayString(5, "Please clear path ahead"); |  |
| 386 |  | playSound(soundBeepBeep); |  |
| 387 |  | } |  |
| 388 |  | ev3StopSound(); |  |
| 389 |  | setDriveTrainSpeed(motor\_power); |  |
| 390 | } |  |  |
| 391 |  |  |  |
| 392 | **void** | **somethingInTheWay** (**int** left\_mot\_pow, **int** right\_mot\_pow) |  |
| 393 | { |  |  |
| 394 |  | // same as apove, just with 2 motor inputs to accomodate the |  |
| 395 |  | // use of this function in turns |  |
| 396 |  | **while**(SensorValue[ULTRASONIC\_PORT] < DIST\_IN\_FRONT\_LIM) |  |

|  |  |  |
| --- | --- | --- |
| 397 | { |  |
| 398 |  | setDriveTrainSpeed(0); |
| 399 |  | eraseDisplay(); |
| 400 |  | displayString(5, "Please clear path ahead"); |
| 401 |  | playSound(soundBeepBeep); |

402

403

404

405

406 }

407

}

ev3StopSound();

motor[LEFT\_MOT\_PORT] = left\_mot\_pow; motor[RIGHT\_MOT\_PORT] = right\_mot\_pow;

408 // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* calculation functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. **int distToDeg**(**float** dist)
2. {
3. // takes a distance and converts it to motor encoder clicks
4. // using wheel radius
5. **return** dist\*180/PI/WHEEL\_RAD;

414 }

415

416 **float degToDist**(**int** deg)

417 {

418 // converts degrees to motor encoder clicks using wheel radius

419 **return** deg\*PI\*WHEEL\_RAD/180;

420 }

421

422 **float average**(**int** value1, **int** value2)

423 {

424 // returns average of two fucntions

425 **return** (abs(value1 + value2)/2.0);

426 }

427

428 // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* movement functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

429 **void setDriveTrainSpeed**(**int** speed)

430 {

431 // accomodates the backwards mounting of drive motors

432 motor[LEFT\_MOT\_PORT] = motor[RIGHT\_MOT\_PORT] = -1\*speed;

433 }

434

1. **void driveDist**(**float** dist, **int** mot\_pow)
2. {
3. // drives specified distance without dropping dominoes
4. setDriveTrainSpeed(mot\_pow);
5. nMotorEncoder[LEFT\_MOT\_PORT] = 0;
6. **while**(abs(nMotorEncoder[LEFT\_MOT\_PORT]) < distToDeg(dist))
7. {
8. // check for break conditions
9. **if**(SensorValue[TOUCH\_PORT])
10. {
11. stopAndKnock();

446 }

447 **else if**(SensorValue[ULTRASONIC\_PORT] < DIST\_IN\_FRONT\_LIM)

448 {

449 somethingInTheWay(mot\_pow);

450 }

451 }

452 setDriveTrainSpeed(0);

453 }

|  |  |  |
| --- | --- | --- |
| 454 |  | |
| 455 | **void driveWhileDropping**(**float** dist, **int** mot\_pow, **bool** &drop\_index, **int** &domino\_count, **float**  &dist\_since\_last\_dom) | |
| 456 | { | |
| 457 | // drives specified distance while dropping dominos at consistent intervals | |
| 458 | setDriveTrainSpeed(mot\_pow); | |
| 459 | nMotorEncoder[LEFT\_MOT\_PORT] = 0; | |
| 460 | nMotorEncoder[RIGHT\_MOT\_PORT] = 0; | |
| 461 | **while**(degToDist(abs(nMotorEncoder(LEFT\_MOT\_PORT))) < dist && domino\_count > 0) | |
| 462 | { | |
| 463 | // check for break conditions | |
| 464 | **if**(SensorValue[TOUCH\_PORT]) | |
| 465 | { | |
| 466 | stopAndKnock(); | |
| 467 | } | |
| 468 | **else if**(SensorValue[ULTRASONIC\_PORT] < DIST\_IN\_FRONT\_LIM) | |
| 469 | { | |
| 470 | somethingInTheWay(mot\_pow); | |
| 471 | } | |
| 472 |  | |
| 473 | // drop domino every DIST\_BETWEEN\_DOMINOS | |
| 474 | **if**(degToDist(abs(nMotorEncoder(RIGHT\_MOT\_PORT))) + dist\_since\_last\_dom >= DIST\_BETWEEN\_DOMINOS) | |
| 475 | { |  |
| 476 | dist\_since\_last\_dom = 0; |  |
| 477 | nMotorEncoder(RIGHT\_MOT\_PORT) = 0; |  |
| 478 | dropDomino(drop\_index, domino\_count); |  |
| 479 | setDriveTrainSpeed(mot\_pow); |  |
| 480 | } |  |
| 481 | } |  |
| 482 | // allows for a smooth transition in the domino path between driving linearly  turning | and |
| 483 | dist\_since\_last\_dom = degToDist(abs(nMotorEncoder(RIGHT\_MOT\_PORT))); |  |
| 484 | } |  |
| 485 |  |  |
| 486 | **void turnInPlace**(**int** angle, **int** mot\_pow) |  |
| 487 | { |  |
| 488 | **int** initialGyro = getGyroDegrees(GYRO\_PORT); |  |
| 489 | **if**(angle < 0) |  |
| 490 | { |  |
| 491 | // turn left |  |
| 492 | motor[LEFT\_MOT\_PORT] = mot\_pow; |  |
| 493 | motor[RIGHT\_MOT\_PORT] = -mot\_pow; |  |
| 494 | **while**(getGyroDegrees(GYRO\_PORT) > initialGyro+angle) |  |
| 495 | { |  |
| 496 | // check for break conditions |  |
| 497 | **if**(SensorValue[TOUCH\_PORT]) |  |
| 498 | { |  |
| 499 | stopAndKnock(); |  |
| 500 | } |  |
| 501 | **else if**(SensorValue[ULTRASONIC\_PORT] < DIST\_IN\_FRONT\_LIM) |  |
| 502 | { |  |
| 503 | somethingInTheWay(mot\_pow, -mot\_pow); |  |
| 504 | } |  |
| 505 | } |  |
| 506 | } |  |
| 507 | **else if**(angle > 0) |  |
| 508 | { |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 509 |  | // turn right | |
| 510 |  | motor[LEFT\_MOT\_PORT] = -mot\_pow; | |
| 511 |  | motor[RIGHT\_MOT\_PORT] = mot\_pow; | |
| 512 |  | **while**(getGyroDegrees(GYRO\_PORT) < initialGyro+angle) | |
| 513 |  | { | |
| 514 |  | // check for break conditions | |
| 515 |  | **if**(SensorValue[TOUCH\_PORT]) | |
| 516 |  | { | |
| 517 |  | stopAndKnock(); | |
| 518 |  | } | |
| 519 |  | **else if**(SensorValue[ULTRASONIC\_PORT] < DIST\_IN\_FRONT\_LIM) | |
| 520 |  | { | |
| 521 |  | somethingInTheWay(-mot\_pow, mot\_pow); | |
| 522 |  | } | |
| 523 |  | } | |
| 524 | } |  | |
| 525 |  |  | |
| 526 |  | setDriveTrainSpeed(0); | |
| 527 | } | | |
| 528 |  | | |
| 529 | **void turnWhileDropping**(**int** angle, **int** speed, **bool** &drop\_index, **int** &domino\_count, **float**  &dist\_since\_last\_dom) | | |
| 530 | { | | |
| 531 | // some concepts taken from: | | |
| 532 | // https://math.stackexchange.com/questions/4310012/calculate-the-turning-radius- turning-circle-of-a-two-wheeled-car | | |
| 533 |  | |  |
| 534 | // turns the robot through a specific radius while dropping dominoes | |  |
| 535 | **float const** TURN\_RATIO = (TURN\_RAD-13.5)/TURN\_RAD; | |  |
| 536 | **int** initialGyro = getGyroDegrees(GYRO\_PORT); | |  |
| 537 | **if**(angle > 0) | |  |
| 538 | { | |  |
| 539 | // turn Right | |  |
| 540 | motor[LEFT\_MOT\_PORT] = -speed; | |  |
| 541 | motor[RIGHT\_MOT\_PORT] = -speed\*TURN\_RATIO; | |  |
| 542 | nMotorEncoder(LEFT\_MOT\_PORT) = 0; | |  |
| 543 | **while**(getGyroDegrees(GYRO\_PORT) < initialGyro+angle && domino\_count | | > 0) |
| 544 | { | |  |
| 545 | // check for break conditions | |  |
| 546 | **if**(SensorValue[TOUCH\_PORT]) | |  |
| 547 | { | |  |
| 548 | stopAndKnock(); | |  |
| 549 | } | |  |
| 550 | **else if**(SensorValue[ULTRASONIC\_PORT] < DIST\_IN\_FRONT\_LIM) | |  |
| 551 | { | |  |
| 552 | somethingInTheWay(-speed, -speed\*TURN\_RATIO); | |  |
| 553 | } | |  |
| 554 |  | |  |
| 555 | >= | **if**(degToDist(abs(nMotorEncoder(LEFT\_MOT\_PORT))) + dist\_since\_last\_dom DIST\_BET\_DOM\_TURNING) | |
| 556 |  | { | |
| 557 |  | // drops domino if correct spacing is reached | |
| 558 |  | dist\_since\_last\_dom = 0; | |
| 559 |  | nMotorEncoder(LEFT\_MOT\_PORT) = 0; | |
| 560 |  | dropDomino(drop\_index, domino\_count); | |
| 561 |  | motor[LEFT\_MOT\_PORT] = -speed; | |
| 562 |  | motor[RIGHT\_MOT\_PORT] = -speed\*TURN\_RATIO; | |
| 563 |  | } | |

|  |  |  |
| --- | --- | --- |
| 564 | } |  |
| 565 | dist\_since\_last\_dom = degToDist(abs(nMotorEncoder(LEFT\_MOT\_PORT))); |
| 566 | } |
| 567 | **else if**(angle < 0) |
| 568 | { |
| 569 | // turn left |
| 570 | motor[LEFT\_MOT\_PORT] = -speed\*TURN\_RATIO; |
| 571 | motor[RIGHT\_MOT\_PORT] = -speed; |
| 572 | nMotorEncoder(RIGHT\_MOT\_PORT) = 0; |
| 573 | **while**(getGyroDegrees(GYRO\_PORT) > initialGyro+angle && domino\_count > | 0) |
| 574 | { |  |
| 575 | // check for break conditions |  |
| 576 | **if**(SensorValue[TOUCH\_PORT]) |  |
| 577 | { |  |
| 578 | stopAndKnock(); |  |
| 579 | } |  |
| 580 | **else if**(SensorValue[ULTRASONIC\_PORT] < DIST\_IN\_FRONT\_LIM) |  |
| 581 | { |  |
| 582 | somethingInTheWay(-speed\*TURN\_RATIO, -speed); |  |
| 583 | } |  |
| 584 | **if**(degToDist(abs(nMotorEncoder(RIGHT\_MOT\_PORT))) + dist\_since\_last\_dom >= DIST\_BET\_DOM\_TURNING) |  |
| 585 | { |  |
| 586 | // drops domino if correct spacing is reached |  |
| 587 | dist\_since\_last\_dom = 0; |  |
| 588 | nMotorEncoder(RIGHT\_MOT\_PORT) = 0; |  |
| 589 | dropDomino(drop\_index, domino\_count); |  |
| 590 | motor[LEFT\_MOT\_PORT] = -speed\*TURN\_RATIO; |  |
| 591 | motor[RIGHT\_MOT\_PORT] = -speed; |  |
| 592 | } |  |
| 593 | } |  |
| 594 | dist\_since\_last\_dom = degToDist(abs(nMotorEncoder(RIGHT\_MOT\_PORT))); |  |
| 595 | } |  |
| 596 | } |  |
| 597 |  |  |
| 598 | **void stopAndKnock**() // Josh |  |
| 599 | { |  |
| 600 | // moves backwards, knocking over first domino |  |
| 601 | nMotorEncoder(LEFT\_MOT\_PORT) = 0; |  |
| 602 | setDriveTrainSpeed(KNOCK\_SPEED); |  |
| 603 | **while**(nMotorEncoder(LEFT\_MOT\_PORT) < distToDeg(DIST\_BETWEEN\_DOMINOS-0.5)) |  |
| 604 | {} |  |
| 605 | setDriveTrainSpeed(0); |  |
| 606 | stopAllTasks(); |  |
| 607 | } |  |
| 608 |  |  |
| 609 | **void openDoor**() // Henrique |  |
| 610 | { |  |
| 611 | motor[DOOR\_MOT\_PORT] = DOOR\_SPEED; |  |
| 612 | **while** (nMotorEncoder(DOOR\_MOT\_PORT)<DOOR\_ANG) |  |
| 613 | { |  |
| 614 | // check for break conditions |  |
| 615 | **if**(SensorValue[TOUCH\_PORT]) |  |
| 616 | { |  |
| 617 | motor[DOOR\_MOT\_PORT] = 0; |  |
| 618 | stopAndKnock(); |  |
| 619 | } |  |
| 620 | } |  |

|  |  |  |
| --- | --- | --- |
| 621 |  | motor[DOOR\_MOT\_PORT] = 0; |
| 622 | } |  |
| 623 |  |  |
| 624 | **void** | **closeDoor**() // Henrique |
| 625 | { |  |
| 626 |  | **if**(!nMotorEncoder(DOOR\_MOT\_PORT)<5) |
| 627 |  | { |
| 628 |  | motor[DOOR\_MOT\_PORT] = -1\*DOOR\_SPEED; |
| 629 |  | **while** (nMotorEncoder(DOOR\_MOT\_PORT)>5) |
| 630 |  | { |
| 631 |  | // check for break conditions |
| 632 |  | **if**(SensorValue[TOUCH\_PORT]) |
| 633 |  | { |
| 634 |  | motor[DOOR\_MOT\_PORT] = 0; |
| 635 |  | stopAndKnock(); |
| 636 |  | } |
| 637 |  | } |
| 638 |  | motor[DOOR\_MOT\_PORT] = 0; |
| 639 |  | } |
| 640 | } |  |