Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Новосибирский государственный технический университет»

Кафедра: Автоматики

**Отчет ПО ПРАКТИКЕ**

Производственная (преддипломная) практика: практика по получению

профессиональных умений и опыта профессиональной деятельности (сосредоточенная)

Направление подготовки: 270304 – Управление в технических системах

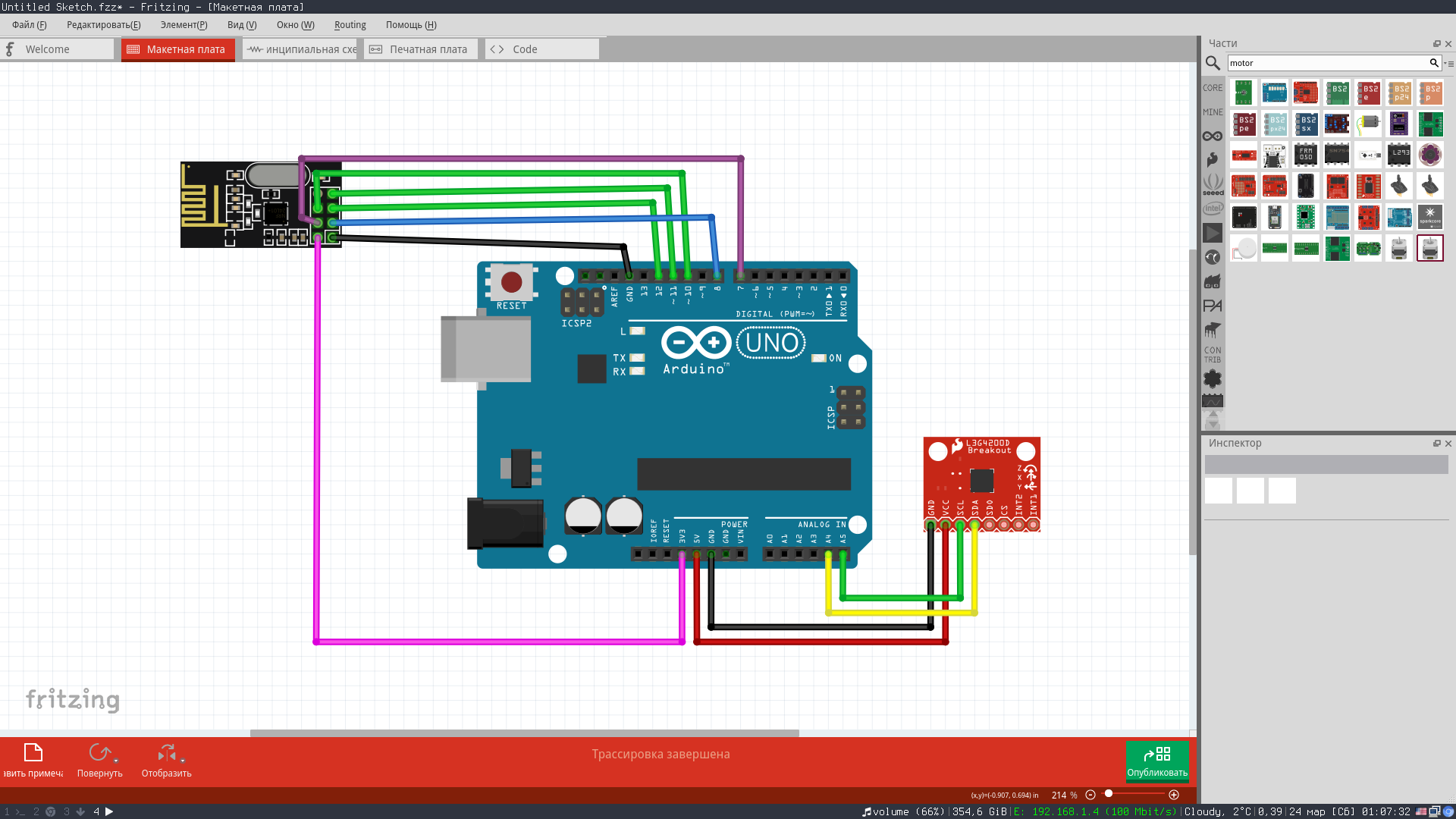
|  |  |
| --- | --- |
| Выполнил:  Студент  Гаевский Илья Валерьевич  (Ф. И. О.)  Группа АА-47  Факультет АВТ.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. | Проверил:  Руководитель от НГТУ Колкер Алексей Борисович  (Ф. И. О.)  Балл: \_\_\_\_\_\_\_\_\_\_, ECTS\_\_\_\_\_\_\_\_\_\_\_\_\_,  Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. |

Новосибирск 2018

**Задание:**

Разработать систему стабилизации мультироторными летательными аппаратами и математическую модель квадрокоптера.

**Ход практики:**

****

*Рис.1* Структурная схема системы.

Компоненты системы:

1. nRF24L01+;
2. MPU9250;
3. Arduino Uno.

В курсовой работе был создан полетный контроллер, принимающий значения от пульта дистанционного управления на основе модуля nRF24L01+, передающий информацию по частоте 2,4 ГГц, сопряженный с Arduino по протоколу SPI. Так же полетный контроллер принимает значения от модуля MPU9250, содержащего в себе акселерометр, гироскоп, магнитометр, барометр, подключенный к Arduino по протоколу I2C. Микроконтроллер, принимает значения от пульта управления о требуемой мощности, тангажу, крену, рысканье; от гироскопа и акселерометра данные о тангаже, крене и рысканьи. На основе этих данных, микроконтроллер высчитывает правило управления и отправляет требуемые значения на моторы. Теперь же, помимо этого надо разработать математическую модель коптера.

**Разработка модели**

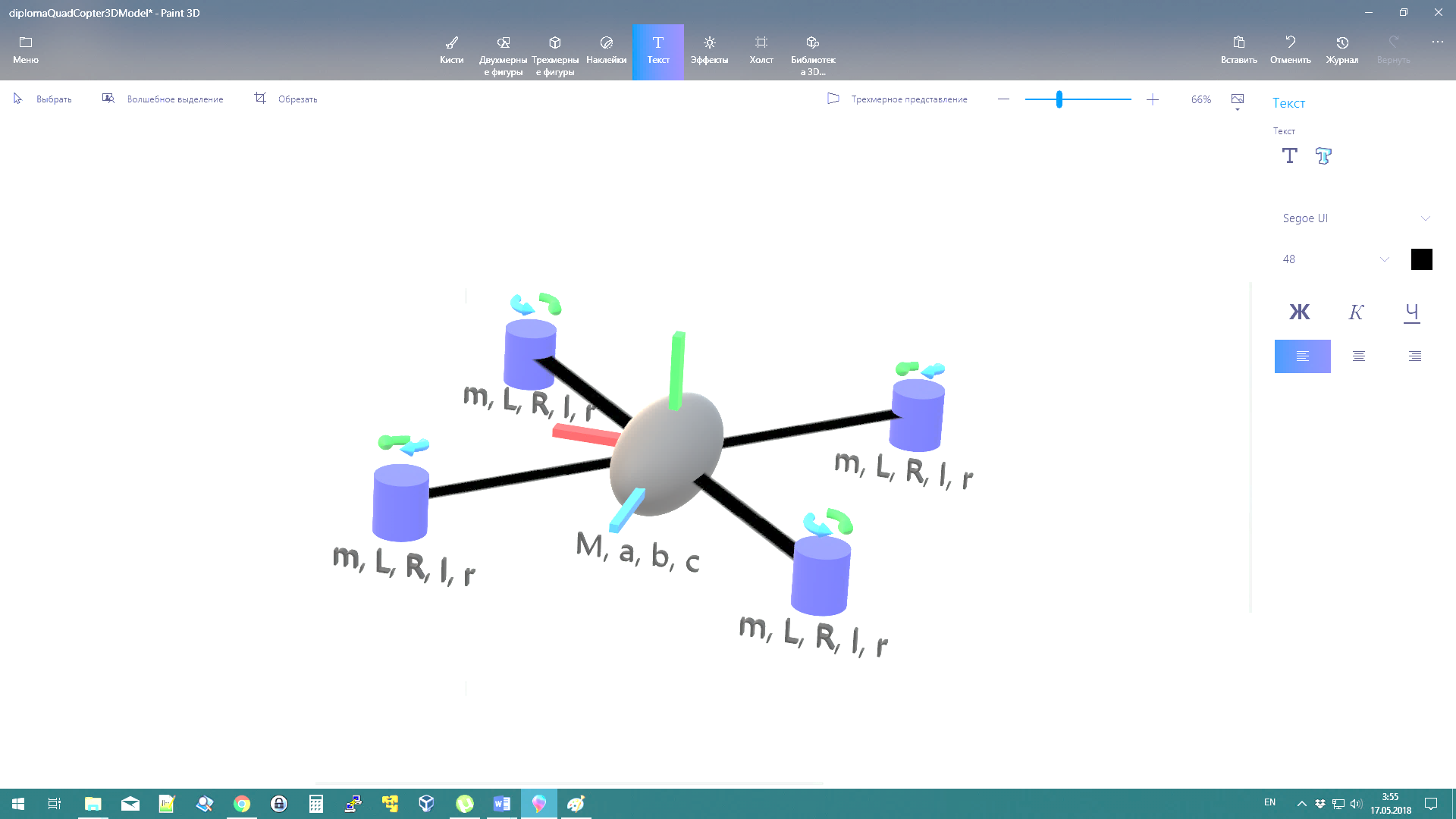


Рис.2. Общий вид коптера

Где ось Ox – голубая, Oy- красная, Oz – зеленая; M – масса тела коптера; a, b, c – параметры элипсоида (по оси Ox – b, Oy – c, Oz – a); m -масса одного двигателя, L – высота двигателя, R – радиус двигателя, l – расстояние от центра коптера до двигателя, r – радиус винта; зелеными стрелками указано направление вращения двигателя, синими – направления момента сопротивления винта, плечи соединения тела и двигателей и тела.

Моменты инерции по осям:

Ox

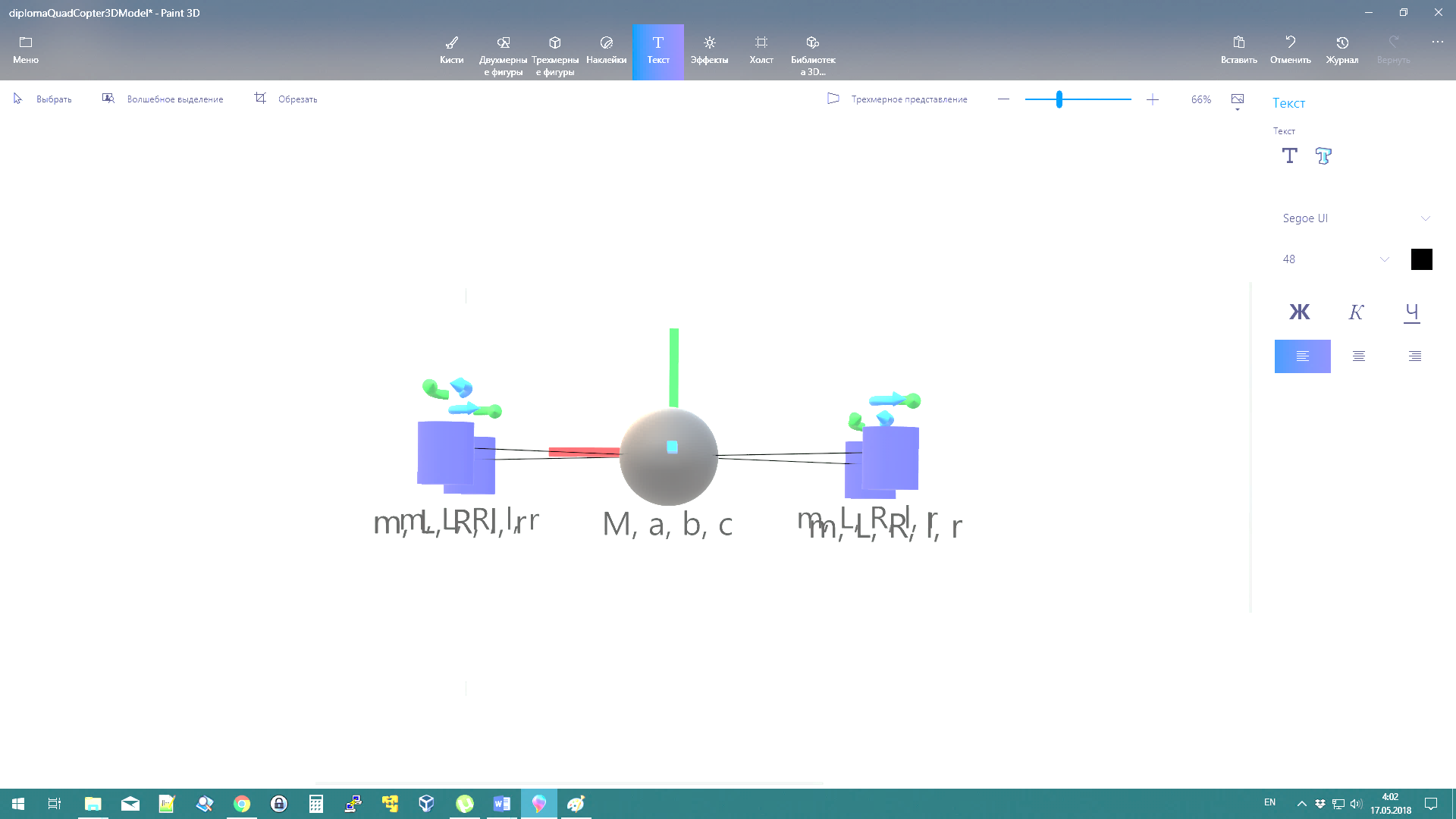


Рис. 3. Модель коптера вдоль оси Ox

Oy

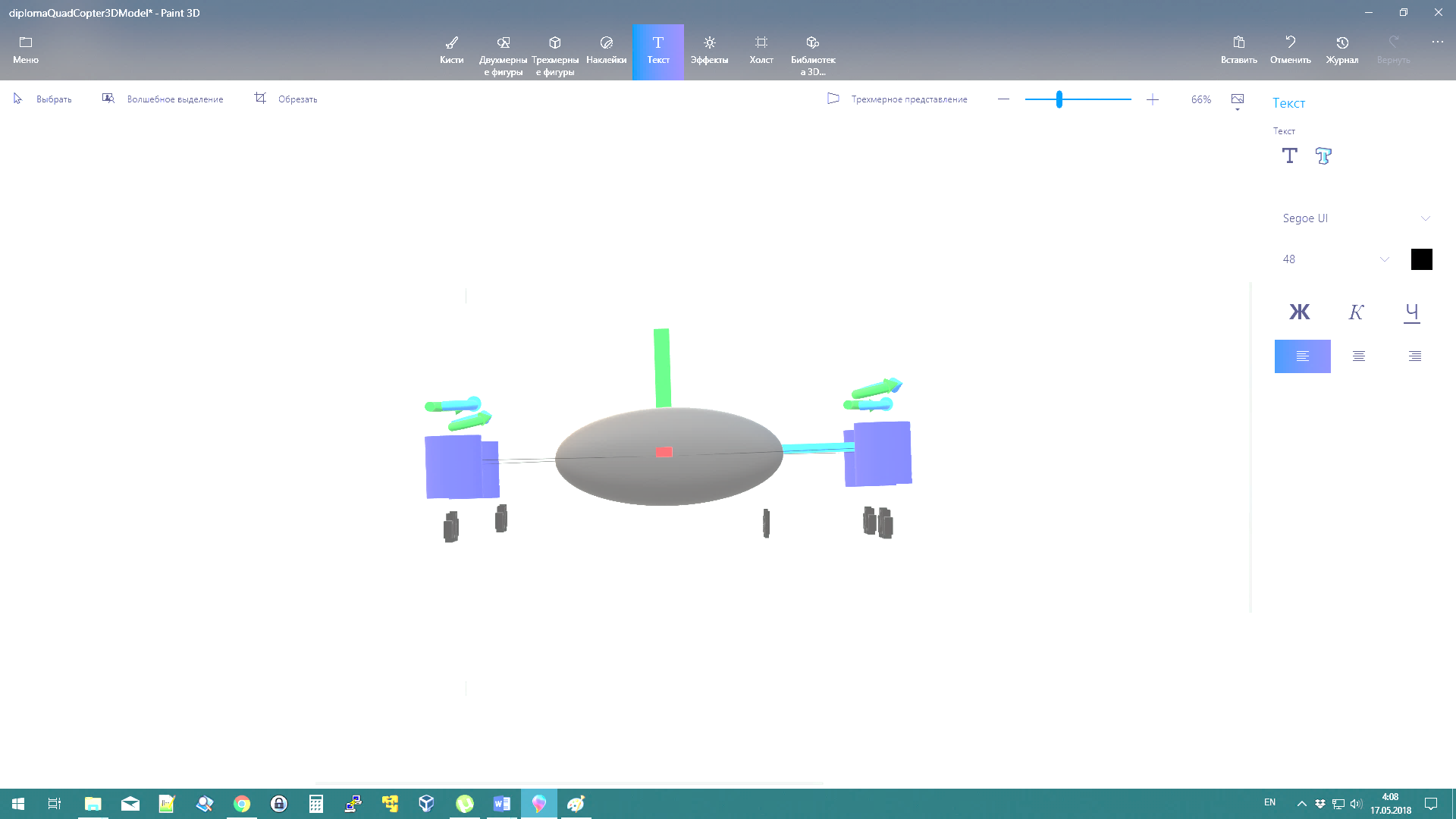


Рис. 4. Модель коптера вдоль оси Oy

Oz

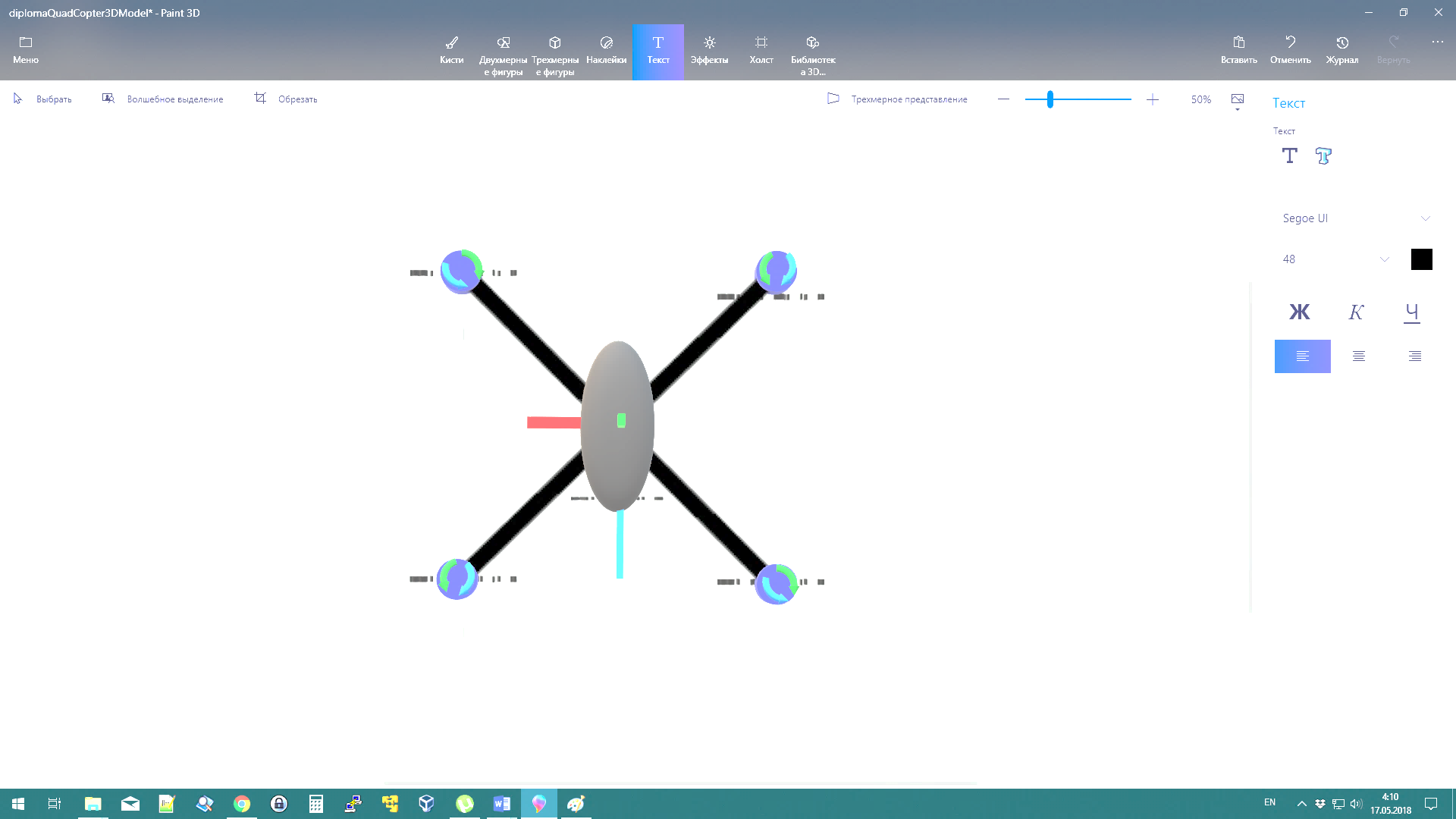


Рис. 5. Модель коптера вдоль оси Oz

Матрицы сил и вращения твердого тела.

Подъемная сила каждого винта с модулем P направлена вверх, сонаправленно оси Oz, относительно самого коптера.

Где, Pk – матрица подъемной силы, разбитой по осям, P – суммарная тяга, – сила тяги *i*-го винта, – коэффициент силы тяги *i*-го винта, – плотность воздуха, – коэффициент подъемной силы, – площадь отметаемой *i*-ым винтом поверхности, - радиус *i*-го винта, *f* – матрица сил сопротивления воздуха, *G* – матрица сил притяжения, *g* – ускорение свободного падения.

Допускаем, что коптер симметричен, центр масс коптера расположен строго в точке начала координат, тогда

Где, MR – матрица результирующего момента, Mq – матрица моментов от винтов kV – коэффициент преобразования напряжения в обороты для напряжений от 1В до 3.7В, где, согласно документации на мотор, идет линейная зависимость.

**Реализация модели в Matlab**

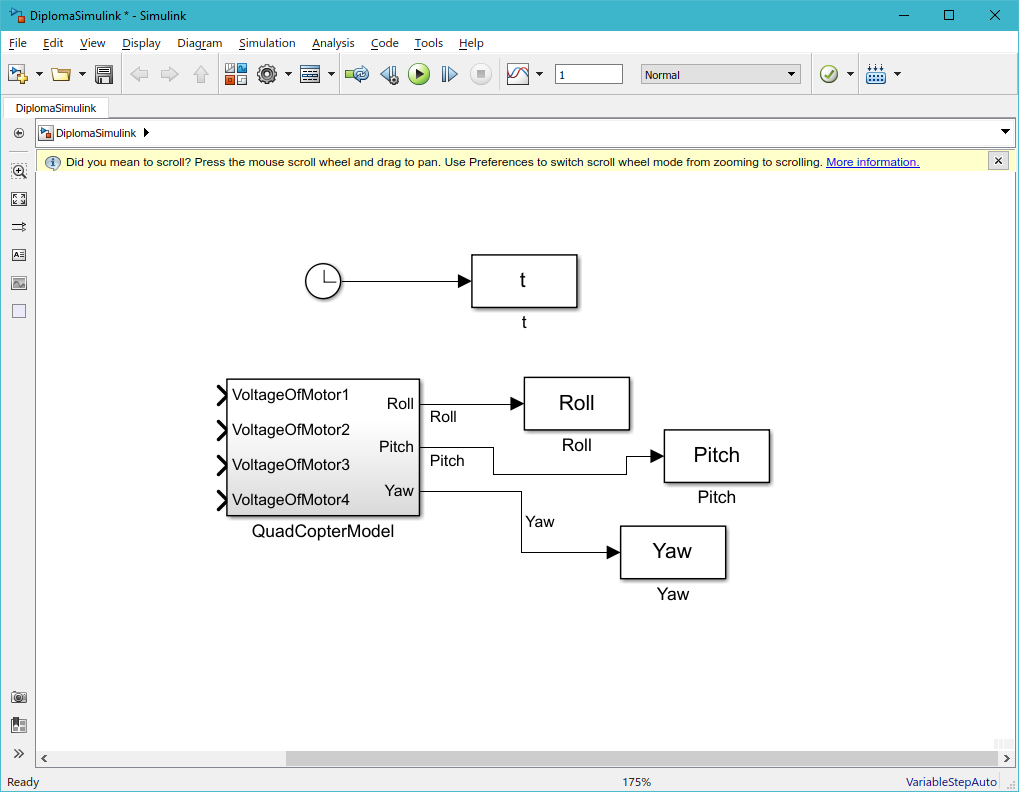


Рис.6. Модель коптера в Simulink

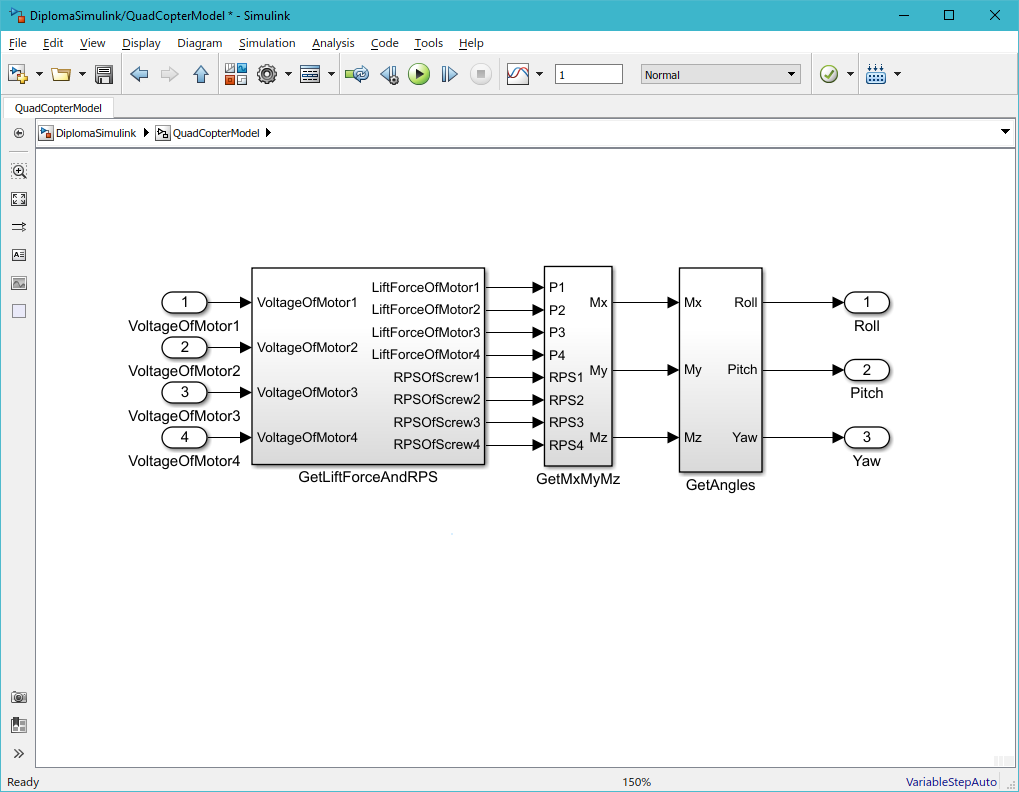


Рис.7. Структурная схема модели коптера

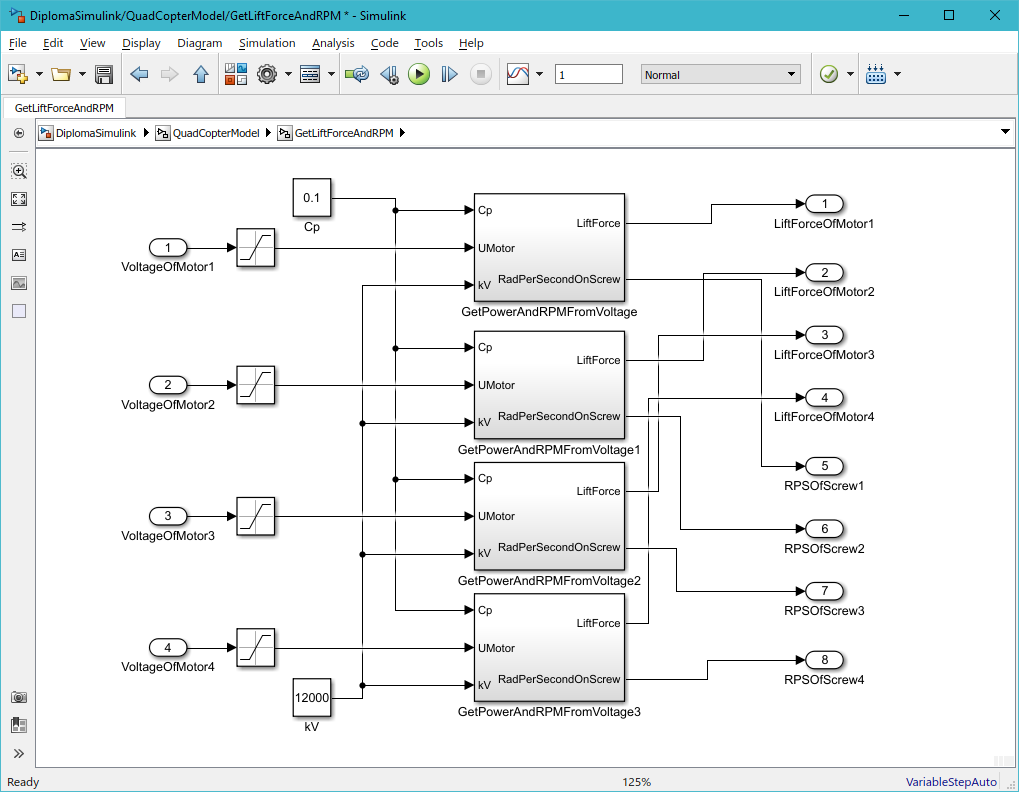


Рис.8. Блок GetLiftForceAndRPS

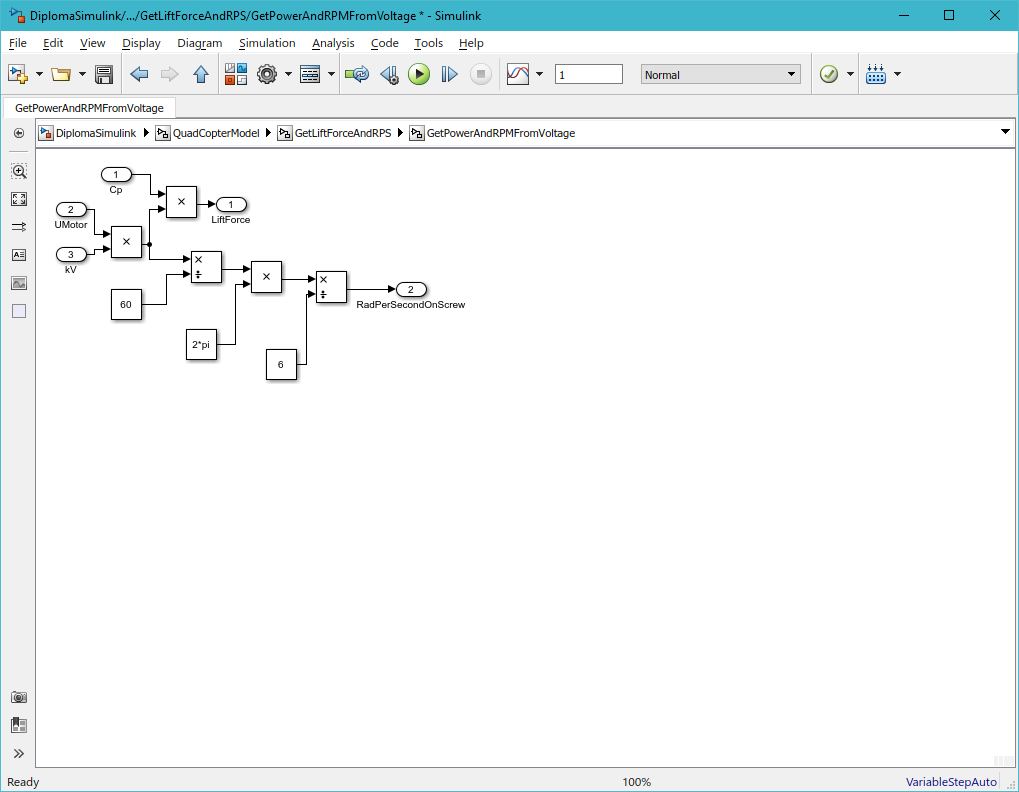


Рис.9. Блок GetPowerAndRPMFromVoltage

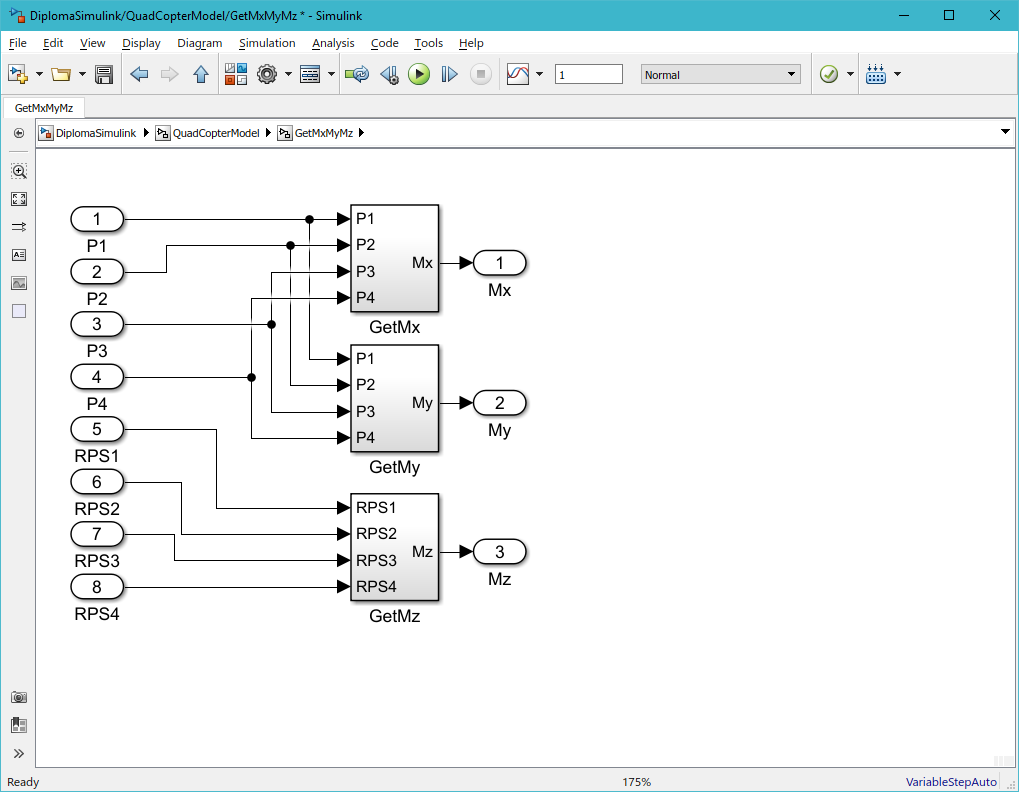


Рис.10. Блок GetMxMyMz

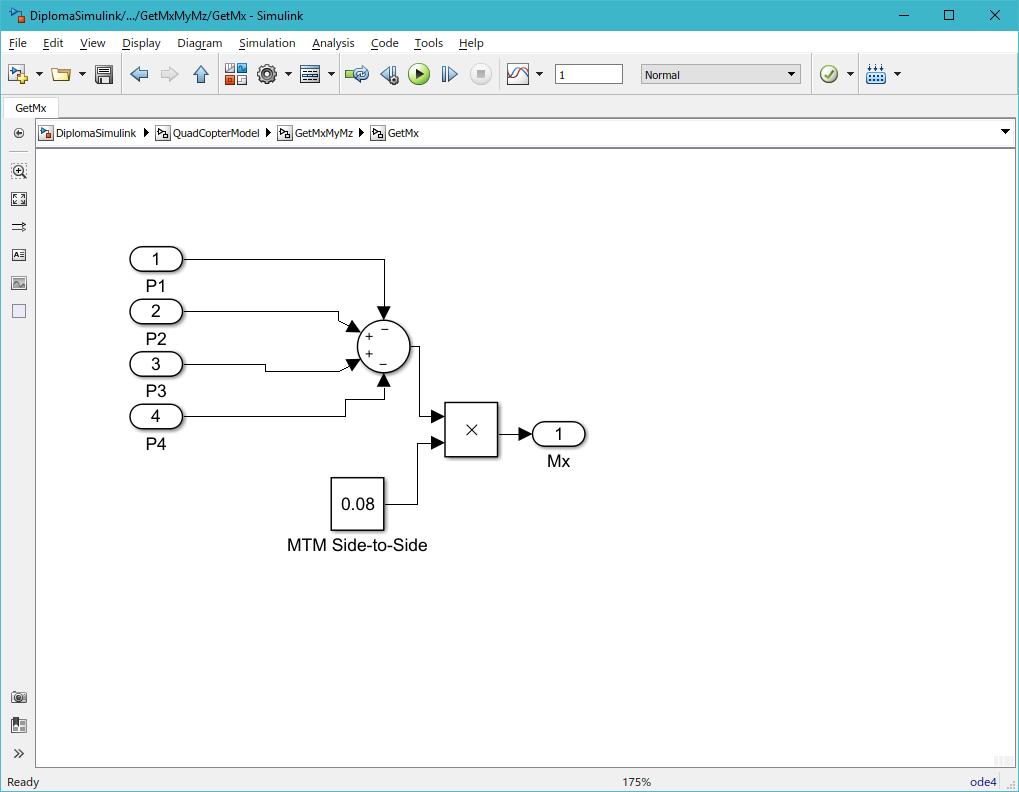


Рис.11. Блок GetMx

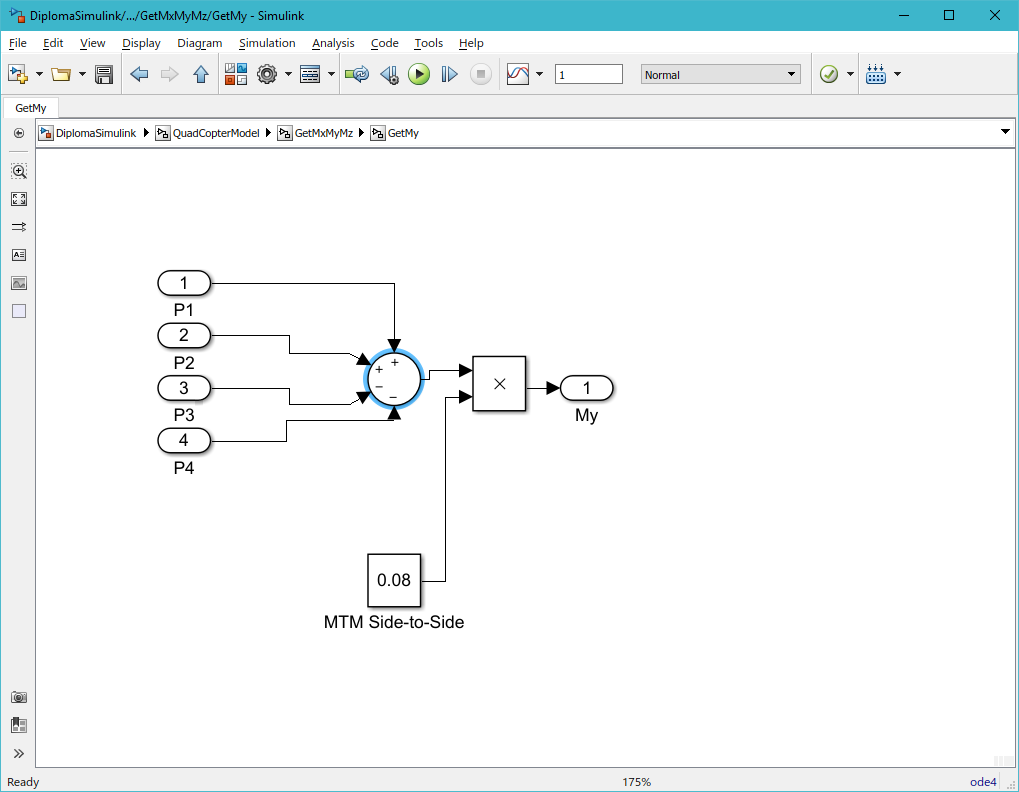


Рис.12. Блок GetMy

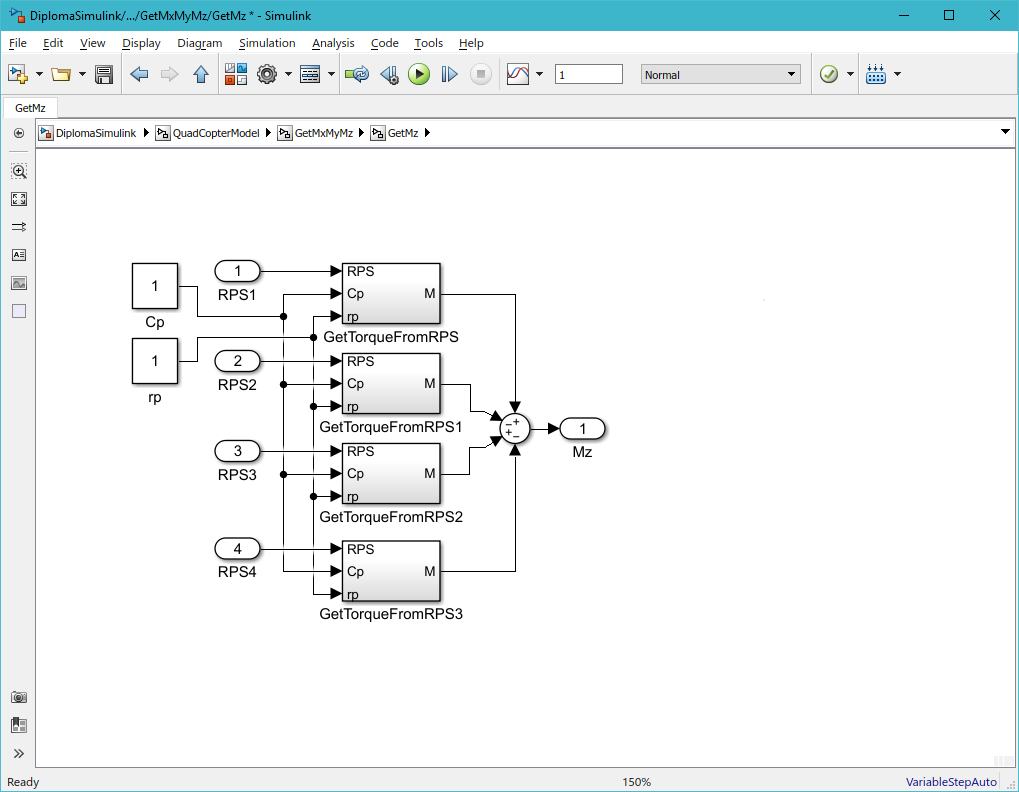


Рис.12. Блок GetMz

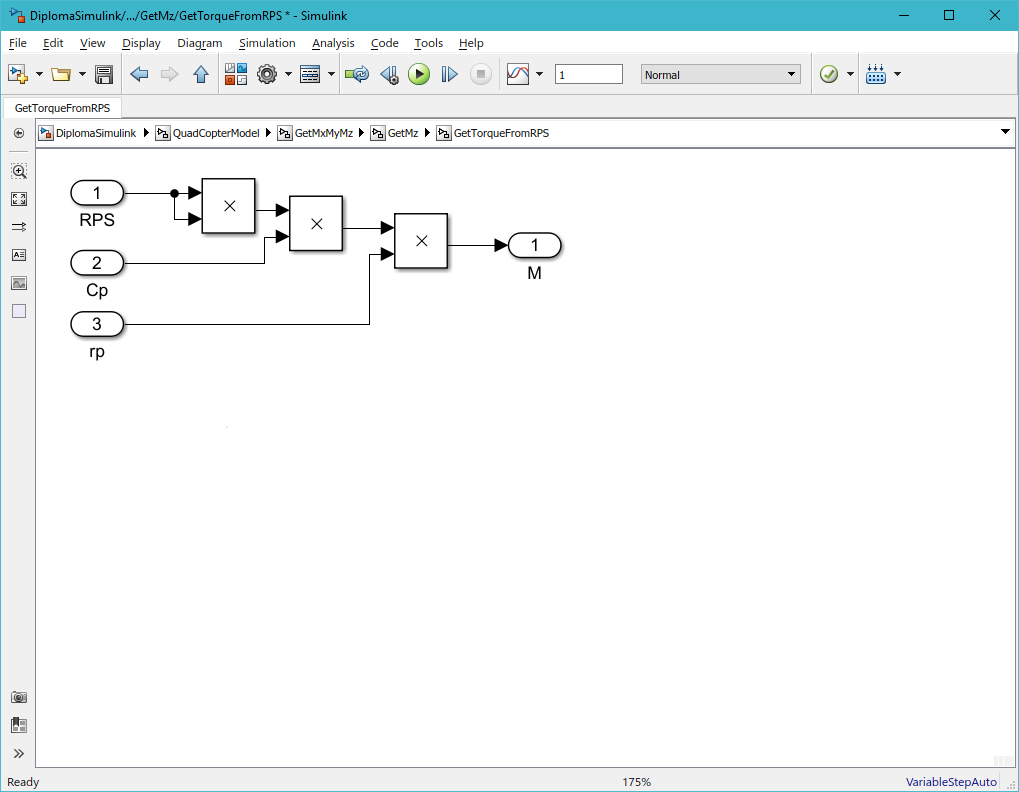


Рис.13. Блок GetTorqueFromRPS

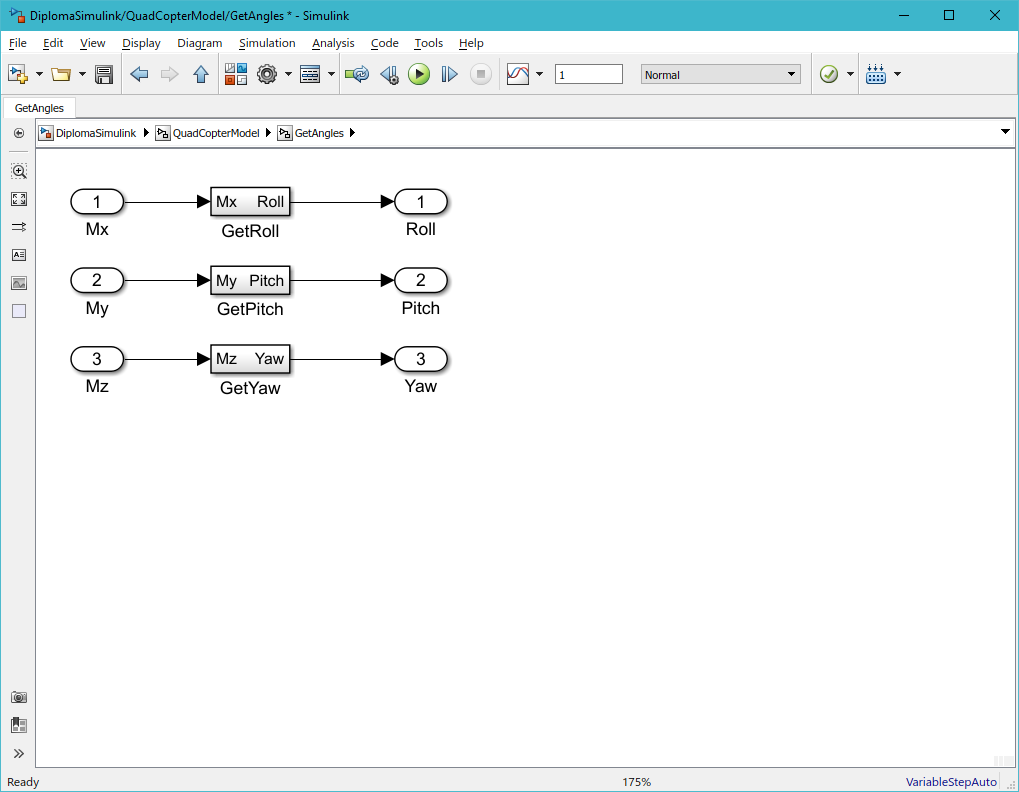


Рис.14. Блок GetAngles

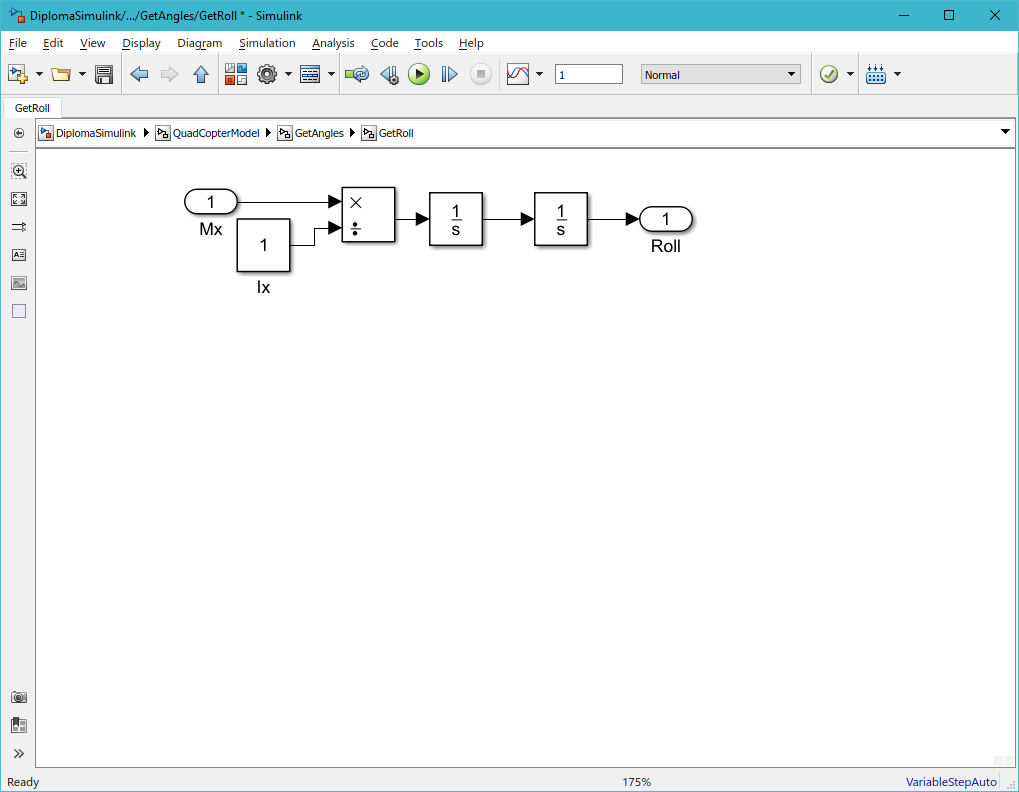


Рис.15. Блок GetRoll

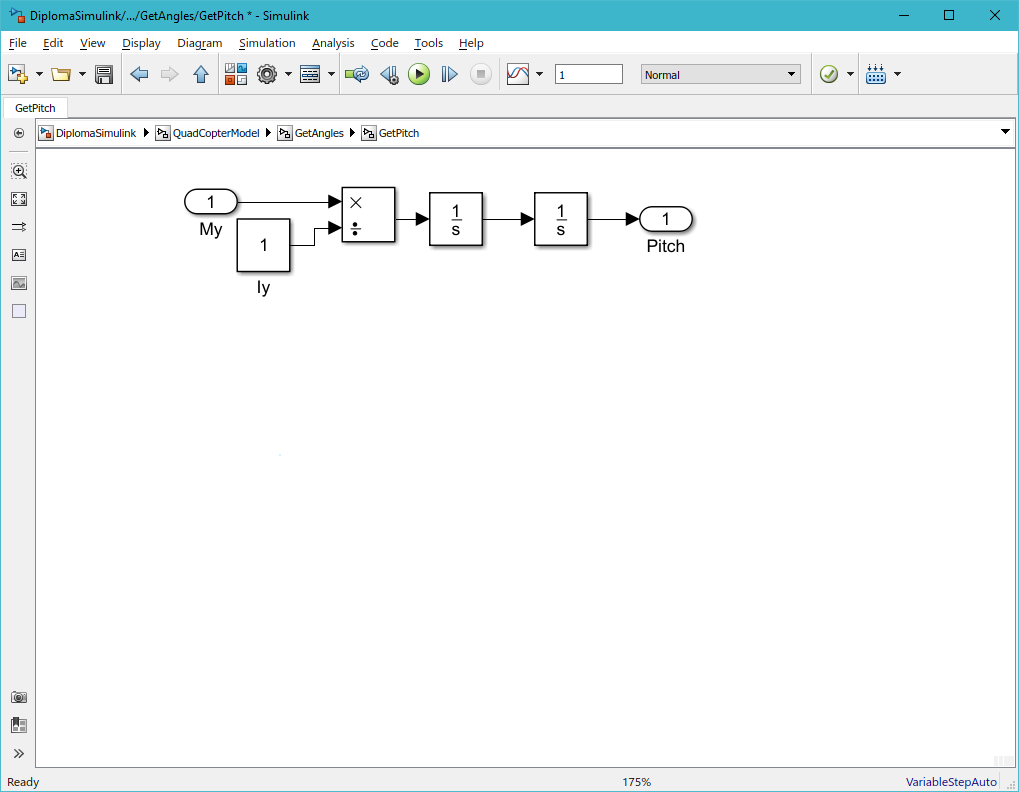


Рис.16. Блок GetPitch

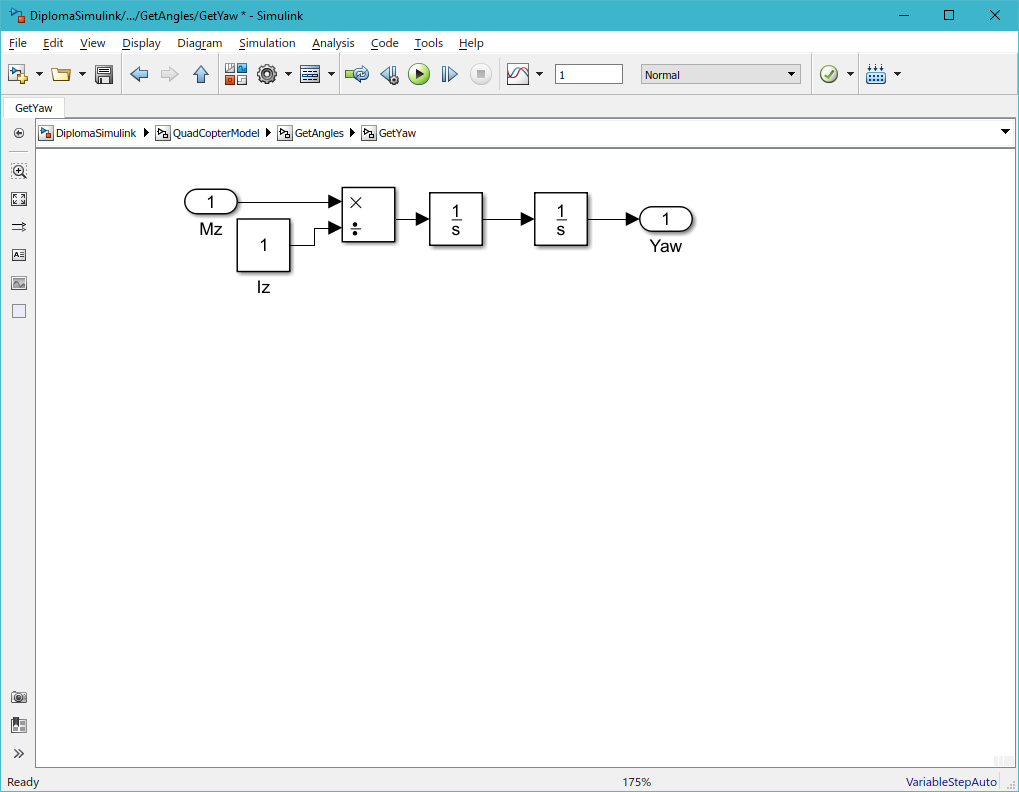


Рис.17. Блок GetYaw

|  |  |  |
| --- | --- | --- |
|  |  |  |

Рис.18-21. Переходный процесс при одинаковом напряжении на всех моторах (взлет)

|  |  |  |
| --- | --- | --- |
|  |  |  |

Рис.22-24. Переходный процесс при большей мощности на передних моторах (тангаж)

|  |  |  |
| --- | --- | --- |
|  |  |  |

Рис.25-27. Переходный процесс при большей мощности на левых моторах (крен)

|  |  |  |
| --- | --- | --- |
|  |  |  |

Рис.28-30. Переходный процесс при большей мощности на диагональных моторах (рысканье)

Как видно из графиков, закон управления для коптера работает оптимально.

**Заключение:**

В ходе данной практики был разработан полетный контроллер с использованием гироскопа, акселерометра, изучена работа с несколькими интерфейсами, построена математической модели квадрокоптера.

**Листинг программы:**

1. #include <printf.h>
3. #include <Servo.h>
5. #include <MPU6050\_tockn.h>
7. #include <Wire.h>
9. #include <RF24.h>
10. #include <RF24\_config.h>
11. #include <nRF24L01.h>
13. #ifdef \_ESP32\_HAL\_I2C\_H\_
14. #define SDA\_PIN 4
15. #define SCL\_PIN 5
16. #endif
18. #define calculatedGyroOffsetX -6.65
19. #define calculatedGyroOffsetY -1.62
20. #define calculatedGyroOffsetZ -1.62
22. #define BAUD\_RATE 19200
24. #define MAX\_SIZE\_OF\_SERIAL 64
26. #define DEBUG false
28. #define ACCELEROMETER\_COEF 0.1
29. #define GYRO\_COEF 0.1
31. **typedef** **enum**
32. {
33. X,
34. Y,
35. Z
36. } COORD;
38. **struct** ANGLE\_COMMAND
39. {
40. COORD coord;
41. **float** angle;
42. };
44. RF24 radio(7, 8);
46. Servo leftFront, rightFront, leftRear, rightRear;
48. MPU6050 mySensor(Wire, ACCELEROMETER\_COEF, GYRO\_COEF);
49. /\*
50. Trottle - высота  (Левый стик, вертикаль)
51. Pitch - Тангаж    (Правый стик, вертикаль)
52. Roll -  Крен      (Правый стик, горизонталь)
53. Yaw -   Рысканье  (Левый стик, горизонталь)
54. \*/
56. //pipes adresses
57. byte addresses[6] = "1Node";
59. byte message[4] = {0, 0, 0, 0};
61. //delaytime
62. **int** dT = 2;
64. //PID's coefficients
65. **float** KpYaw = 1.0, KiYaw = 1.0, KdYaw = 1.0;       //YAW   РЫСКАНЬЕ - Z
66. **float** KpRoll = 1.0, KiRoll = 1.0, KdRoll = 1.0;    //ROLL  КРЕН     - Y
67. **float** KpPitch = 1.0, KiPitch = 1.0, KdPitch = 1.0; //PITCH ТАНГАЖ   - X
69. //PID's values to storage
70. **int** PIDYaw = 0, PIDRoll = 0, PIDPitch = 0;
72. //Errors to PID
73. **int** errorYaw = 0, errorRoll = 0, errorPitch = 0;
75. //Last values of calculateErrors for derivation
76. **int** lastYaw = 0, lastRoll = 0, lastPitch = 0;
78. //Integrator values for inegration;
79. **int** integratedYaw = 0, integratedRoll = 0, integratedPitch = 0;
81. //Values for motors
82. **int** LF, RF, LR, RR;
84. //Input values from transmitter
85. **int** inTrottle = 0, inYaw = 0, inRoll = 0, inPitch = 0;
87. //Input raw values from Gyro
88. **float** inGyroX = 0.0, inGyroY = 0.0, inGyroZ = 0.0;
90. //Input raw values from Accel
91. **float** inAccX = 0.0, inAccY = 0.0, inAccZ = 0.0;
93. //Input raw values from AccelAngle
94. **float** inAccAngleX = 0.0, inAccAngleY = 0.0;
96. //Input raw values from GyroAngle
97. **float** inGyroAngleX = 0.0, inGyroAngleY = 0.0, inGyroAngleZ = 0.0;
99. //Input raw values from Angle
100. **float** inAngleX = 0.0, inAngleY = 0.0, inAngleZ = 0.0;
102. //Input filtered values from Gyro
103. **float** filteredGyroYaw = 0.0, filteredGyroRoll = 0.0, filteredGyroPitch = 0.0;
105. //Input filtered values from Accel
106. **float** filteredAccYaw = 0.0, filteredAccRoll = 0.0, filteredAccPitch = 0.0;
108. //Variables for Kalman's filter
109. **float** deviationGyroYaw = 0.0, deviationGyroRoll = 0.0, deviationGyroPitch = 0.0; //middle deviation
110. **float** speedGyroYaw = 0.0, speedGyroRoll = 0.0, speedGyroPitch = 0.0;             //speed of working
112. **float** PcGyroYaw = 0.0, PcGyroRoll = 0.0, PcGyroPitch = 0.0;
113. **float** GGyroYaw = 0.0, GGyroRoll = 0.0, GGyroPitch = 0.0;
114. **float** PGyroYaw = 0.0, PGyroRoll = 0.0, PGyroPitch = 0.0;
116. **float** deviationAccYaw = 0.0, deviationAccRoll = 0.0, deviationAccPitch = 0.0;
117. **float** speedAccYaw = 0.0, speedAccRoll = 0.0, speedAccPitch = 0.0;
119. **float** PcAccYaw = 0.0, PcAccRoll = 0.0, PcAccPitch = 0.0;
120. **float** GAccYaw = 0.0, GAccRoll = 0.0, GAccPitch = 0.0;
121. **float** PAccYaw = 0.0, PAccRoll = 0.0, PAccPitch = 0.0;
123. **void** writeMotors()
124. {
125. leftFront.write(LF);
126. rightFront.attach(RF);
127. leftRear.attach(LR);
128. rightRear.attach(RR);
129. }
131. **void** getGyro()
132. {
133. inGyroX = mySensor.getGyroX();
134. inGyroY = mySensor.getGyroY();
135. inGyroZ = mySensor.getGyroZ();
136. }
138. **void** getAccel()
139. {
140. inAccX = mySensor.getAccX();
141. inAccY = mySensor.getAccY();
142. inAccZ = mySensor.getAccZ();
143. }
145. **void** getAccAngles()
146. {
147. inAccAngleX = mySensor.getAccAngleX();
148. inAccAngleY = mySensor.getAccAngleY();
149. }
151. **void** getGyroAngles()
152. {
153. inGyroAngleX = mySensor.getGyroAngleX();
154. inGyroAngleY = mySensor.getGyroAngleY();
155. inGyroAngleZ = mySensor.getGyroAngleZ();
156. }
158. **void** getAngles()
159. {
160. inAngleX = mySensor.getAngleX();
161. inAngleY = mySensor.getAngleY();
162. inAngleZ = mySensor.getAngleZ();
163. }
165. **void** getData()
166. {
167. radio.read(&message, **sizeof**(message));
168. inTrottle = message[0];
169. inYaw = message[1] - 512;
170. inRoll = message[2] - 512;
171. inPitch = message[3] - 512;
172. }
174. **void** filterGyro()
175. {
176. PcGyroYaw = PGyroYaw + speedGyroYaw;
177. GGyroYaw = PcGyroYaw / (PcGyroYaw + deviationGyroYaw);
178. PGyroYaw = (1 - GGyroYaw) \* PcGyroYaw;
179. filteredGyroYaw = GGyroYaw \* (inGyroZ - filteredGyroYaw) + filteredGyroYaw;
181. PcGyroRoll = PGyroRoll + speedGyroRoll;
182. GGyroRoll = PcGyroRoll / (PcGyroRoll + deviationGyroRoll);
183. PGyroRoll = (1 - GGyroRoll) \* PcGyroRoll;
184. filteredGyroRoll = GGyroRoll \* (inGyroX - filteredGyroRoll) + filteredGyroRoll;
186. PcGyroPitch = PGyroPitch + speedGyroPitch;
187. GGyroPitch = PcGyroPitch / (PcGyroPitch + deviationGyroPitch);
188. PGyroPitch = (1 - GGyroPitch) \* PcGyroPitch;
189. filteredGyroPitch = GGyroPitch \* (inGyroY - filteredGyroPitch) + filteredGyroPitch;
190. }
192. **void** filterAccel()
193. {
194. PcAccYaw = PAccYaw + speedAccYaw;
195. GAccYaw = PcAccYaw / (PcAccYaw + deviationAccYaw);
196. PAccYaw = (1 - GAccYaw) \* PcAccYaw;
197. filteredAccYaw = GAccYaw \* (inAccZ - filteredAccYaw) + filteredAccYaw;
199. PcAccRoll = PAccRoll + speedAccRoll;
200. GAccRoll = PcAccRoll / (PcAccRoll + deviationAccRoll);
201. PAccRoll = (1 - GAccRoll) \* PcAccRoll;
202. filteredAccRoll = GAccRoll \* (inAccX - filteredAccRoll) + filteredAccRoll;
204. PcAccPitch = PAccPitch + speedAccPitch;
205. GAccPitch = PcAccPitch / (PcAccPitch + deviationAccPitch);
206. PAccPitch = (1 - GAccPitch) \* PcAccPitch;
207. filteredAccPitch = GAccPitch \* (inAccY - filteredAccPitch) + filteredAccPitch;
208. }
210. **void** PIDs()
211. {
212. PIDYaw = (**int**)(KpYaw \* errorYaw + KdYaw \* (errorYaw - lastYaw) + KiYaw \* (integratedYaw + errorYaw));
213. integratedYaw += errorYaw;
214. lastYaw = errorYaw;
216. PIDRoll = (**int**)(KpRoll \* errorRoll + KdRoll \* (errorRoll - lastRoll) + KiRoll \* (integratedRoll + errorRoll));
217. integratedRoll += errorRoll;
218. lastRoll = errorRoll;
220. PIDPitch = (**int**)(KpPitch \* errorPitch + KdPitch \* (errorPitch - lastPitch) + KiPitch \* (integratedPitch + errorPitch));
221. integratedPitch += errorPitch;
222. lastPitch = errorPitch;
223. }
225. **void** calculateErrors()
226. {
227. errorYaw = inYaw - filteredGyroYaw;
228. errorRoll = inRoll - filteredGyroRoll;
229. errorPitch = inPitch - filteredGyroPitch;
230. }
232. **void** countMotors()
233. {
234. LF = inTrottle - PIDPitch - PIDYaw + PIDRoll;
235. RF = inTrottle - PIDPitch + PIDYaw - PIDRoll;
236. LR = inTrottle + PIDPitch + PIDYaw + PIDRoll;
237. RR = inTrottle + PIDPitch - PIDYaw - PIDRoll;
238. }
240. **void** readCommandFromSerial()
241. {
242. **float** angle = 0.0;
243. COORD inputCoord;
244. **char** message[MAX\_SIZE\_OF\_SERIAL];
245. **int** i = 0;
247. **for** (i = 0; i < MAX\_SIZE\_OF\_SERIAL; i++)
248. {
249. message[i] = '\0';
250. }
252. i = 0;
253. **while** (Serial.available() > 0 && i < MAX\_SIZE\_OF\_SERIAL)
254. {
255. message[i] = Serial.read();
256. i++;
257. }
258. **if** (i == 0)
259. **return**;
261. **if** (strncmp(message, "Z", 1) == 0 || strncmp(message, "z", 1) == 0)
262. inputCoord = Z;
263. **else** **if** (strncmp(message, "Y", 1) == 0 || strncmp(message, "y", 1) == 0)
264. inputCoord = Y;
265. **else** **if** (strncmp(message, "X", 1) == 0 || strncmp(message, "x", 1) == 0)
266. inputCoord = X;
268. **char** \*pch = strtok(message, " ");
269. pch = strtok(NULL, " ");
271. Serial.print("\nDEBUG READED:");
272. Serial.println("\tmessage");
273. Serial.println(message);
275. Serial.println("\tpch");
276. Serial.println(pch);
278. Serial.print("\tinputCoord:");
279. Serial.print(inputCoord);
280. Serial.print("\tAngle:");
281. Serial.print(angle);
282. Serial.flush();
284. // switch (inputCoord)
285. // {
286. // case Z:
287. //   inYaw = angle;
288. //   break;
289. // case Y:
290. //   inRoll = angle;
291. //   break;
292. // case X:
293. //   inPitch = angle;
294. //   break;
295. // default:
296. //   break;
297. // }
298. }
300. ANGLE\_COMMAND parseCommandFromSerial()
301. {
302. **struct** ANGLE\_COMMAND temp;
303. temp.coord = Z;
304. temp.angle = 10;
305. **return** temp;
306. }
308. **void** setup()
309. {
310. Serial.begin(BAUD\_RATE);
311. // put your setup code here, to run once:
312. // setup pins (digital: 2 in for axelerometer and gyro, 2 in transmitter, 4 out for motors)
313. leftFront.attach(2);
314. rightFront.attach(3);
315. leftRear.attach(4);
316. rightRear.attach(5);
318. #ifdef \_ESP32\_HAL\_I2C\_H\_        // For ESP32
319. Wire.begin(SDA\_PIN, SCL\_PIN); // SDA, SCL
320. #else
321. Wire.begin();
322. #endif
324. mySensor.begin();
326. **if** (DEBUG)
327. mySensor.calcGyroOffsets(**true**);
328. **else**
329. {
330. mySensor.calcGyroOffsets();
331. mySensor.setGyroOffsets(calculatedGyroOffsetX, calculatedGyroOffsetY, calculatedGyroOffsetZ);
332. }
334. radio.begin();
335. radio.openReadingPipe(1, addresses);
336. radio.startListening();
337. }
339. **void** debugOutput(**const** **char** \*param)
340. {
341. **if** (strcmp(param, "angle") == 0)
342. {
343. Serial.print("\nangleX:");
344. Serial.print(inAngleX);
346. Serial.print("\tangleY:");
347. Serial.print(inAngleY);
349. Serial.print("\tangleZ:");
350. Serial.print(inAngleZ);
351. **return**;
352. }
353. }
355. **void** loop()
356. {
357. // put your main code here, to run repeatedly:
358. //Get Datas
360. **if** (radio.available())
361. getData();
363. //if (DEBUG)
364. //{
365. readCommandFromSerial();
366. //}
368. mySensor.update();
369. getGyro();
370. getAccel();
371. getGyroAngles();
372. getAccAngles();
373. getAngles();
375. //debugOutput("angle");
377. //filter in
378. filterGyro();
379. filterAccel();
380. //calculate calculateErrors
381. calculateErrors();
382. //PID
383. PIDs();
384. //count motors
385. countMotors();
386. //write motors
387. writeMotors();
389. delay(dT);
390. }

**Список литературы:**

1. <https://www.lucidar.me/en/inertial-measurement-unit/mpu-9250-and-arduino-9-axis-imu/>

Описание работы с модулем MPU9250

1. <http://docs.cntd.ru/document/gost-20058-80>

Основы динамики летательных аппаратов в атмосфере

1. <https://geektimes.ru/post/258196/>

Создание квадрокоптера