МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Автоматики

Отчет по преддипломной практике

Факультет: АВТ

Группа: АА-47

Студент: Гаевский И.В. Преподаватель: к.т.н. Колкер А.Б.

Дата выполнения:

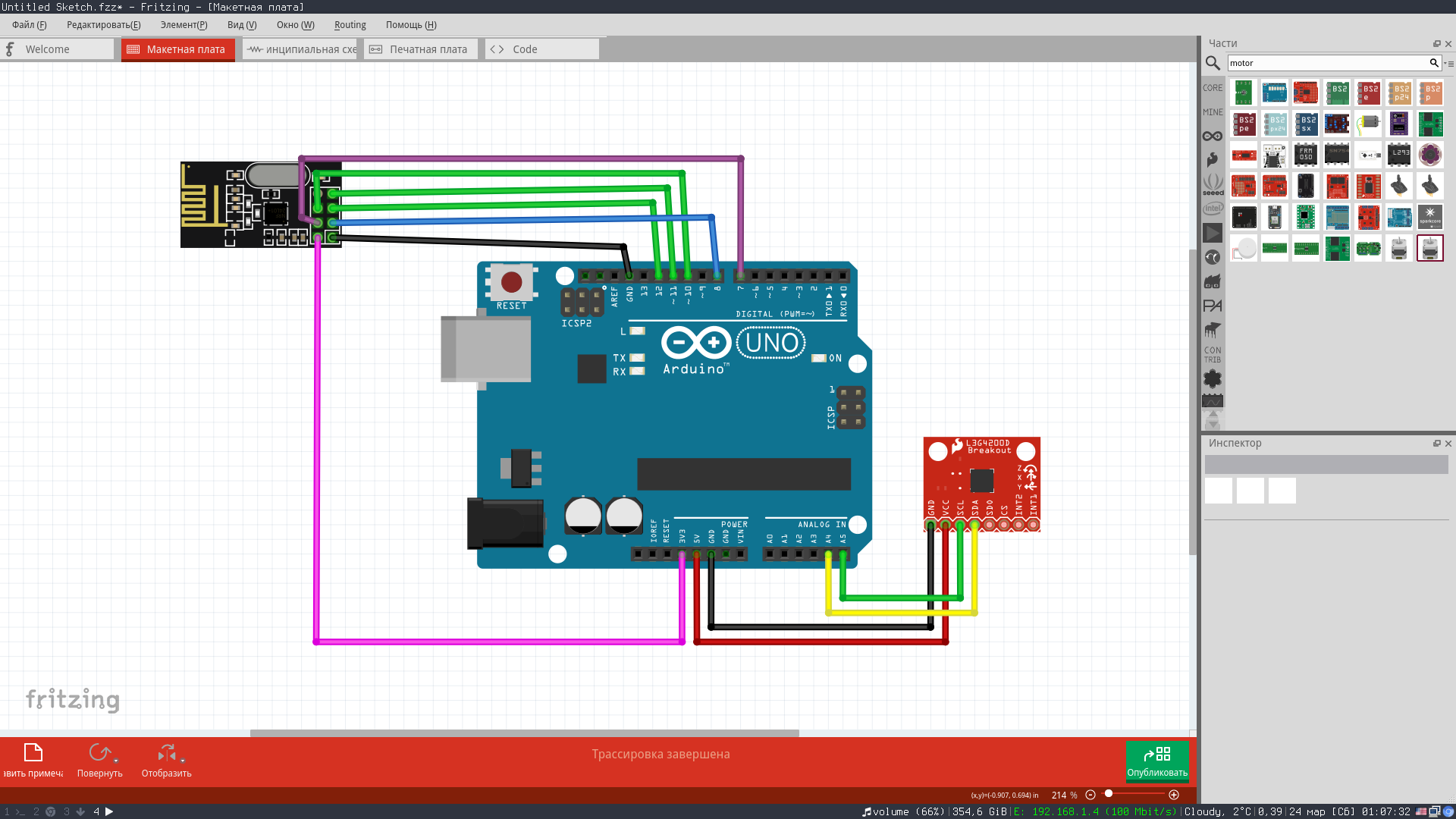
Отметка о защите:

Новосибирск, 2018 г.

**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.

**Листинг программы:**

|  |
| --- |
| #include <printf.h> |
|  |  |
|  | #include <Servo.h> |
|  |  |
|  | #include <MPU6050\_tockn.h> |
|  |  |
|  | #include <Wire.h> |
|  |  |
|  | #include <RF24.h> |
|  | #include <RF24\_config.h> |
|  | #include <nRF24L01.h> |
|  |  |
|  | #ifdef \_ESP32\_HAL\_I2C\_H\_ |
|  | #define SDA\_PIN 4 |
|  | #define SCL\_PIN 5 |
|  | #endif |
|  |  |
|  | #define calculatedGyroOffsetX -6.65 |
|  | #define calculatedGyroOffsetY -1.62 |
|  | #define calculatedGyroOffsetZ -1.62 |
|  |  |
|  | #define BAUD\_RATE 19200 |
|  |  |
|  | #define MAX\_SIZE\_OF\_SERIAL 64 |
|  |  |
|  | #define DEBUG false |
|  |  |
|  | #define ACCELEROMETER\_COEF 0.1 |
|  | #define GYRO\_COEF 0.1 |
|  |  |
|  | typedef enum |
|  | { |
|  | X, |
|  | Y, |
|  | Z |
|  | } COORD; |
|  |  |
|  | struct ANGLE\_COMMAND |
|  | { |
|  | COORD coord; |
|  | float angle; |
|  | }; |
|  |  |
|  | RF24 radio(7, 8); |
|  |  |
|  | Servo leftFront, rightFront, leftRear, rightRear; |
|  |  |
|  | MPU6050 mySensor(Wire, ACCELEROMETER\_COEF, GYRO\_COEF); |
|  | /\* |
|  | Trottle - высота (Левый стик, вертикаль) |
|  | Pitch - Тангаж (Правый стик, вертикаль) |
|  | Roll - Крен (Правый стик, горизонталь) |
|  | Yaw - Рысканье (Левый стик, горизонталь) |
|  | \*/ |
|  |  |
|  | //pipes adresses |
|  | byte addresses[6] = "1Node"; |
|  |  |
|  | byte message[4] = {0, 0, 0, 0}; |
|  |  |
|  | //delaytime |
|  | int dT = 2; |
|  |  |
|  | //PID's coefficients |
|  | float KpYaw = 1.0, KiYaw = 1.0, KdYaw = 1.0; //YAW РЫСКАНЬЕ - Z |
|  | float KpRoll = 1.0, KiRoll = 1.0, KdRoll = 1.0; //ROLL КРЕН - Y |
|  | float KpPitch = 1.0, KiPitch = 1.0, KdPitch = 1.0; //PITCH ТАНГАЖ - X |
|  |  |
|  | //PID's values to storage |
|  | int PIDYaw = 0, PIDRoll = 0, PIDPitch = 0; |
|  |  |
|  | //Errors to PID |
|  | int errorYaw = 0, errorRoll = 0, errorPitch = 0; |
|  |  |
|  | //Last values of calculateErrors for derivation |
|  | int lastYaw = 0, lastRoll = 0, lastPitch = 0; |
|  |  |
|  | //Integrator values for inegration; |
|  | int integratedYaw = 0, integratedRoll = 0, integratedPitch = 0; |
|  |  |
|  | //Values for motors |
|  | int LF, RF, LR, RR; |
|  |  |
|  | //Input values from transmitter |
|  | int inTrottle = 0, inYaw = 0, inRoll = 0, inPitch = 0; |
|  |  |
|  | //Input raw values from Gyro |
|  | float inGyroX = 0.0, inGyroY = 0.0, inGyroZ = 0.0; |
|  |  |
|  | //Input raw values from Accel |
|  | float inAccX = 0.0, inAccY = 0.0, inAccZ = 0.0; |
|  |  |
|  | //Input raw values from AccelAngle |
|  | float inAccAngleX = 0.0, inAccAngleY = 0.0; |
|  |  |
|  | //Input raw values from GyroAngle |
|  | float inGyroAngleX = 0.0, inGyroAngleY = 0.0, inGyroAngleZ = 0.0; |
|  |  |
|  | //Input raw values from Angle |
|  | float inAngleX = 0.0, inAngleY = 0.0, inAngleZ = 0.0; |
|  |  |
|  | //Input filtered values from Gyro |
|  | float filteredGyroYaw = 0.0, filteredGyroRoll = 0.0, filteredGyroPitch = 0.0; |
|  |  |
|  | //Input filtered values from Accel |
|  | float filteredAccYaw = 0.0, filteredAccRoll = 0.0, filteredAccPitch = 0.0; |
|  |  |
|  | //Variables for Kalman's filter |
|  | float deviationGyroYaw = 0.0, deviationGyroRoll = 0.0, deviationGyroPitch = 0.0; //middle deviation |
|  | float speedGyroYaw = 0.0, speedGyroRoll = 0.0, speedGyroPitch = 0.0; //speed of working |
|  |  |
|  | float PcGyroYaw = 0.0, PcGyroRoll = 0.0, PcGyroPitch = 0.0; |
|  | float GGyroYaw = 0.0, GGyroRoll = 0.0, GGyroPitch = 0.0; |
|  | float PGyroYaw = 0.0, PGyroRoll = 0.0, PGyroPitch = 0.0; |
|  |  |
|  | float deviationAccYaw = 0.0, deviationAccRoll = 0.0, deviationAccPitch = 0.0; |
|  | float speedAccYaw = 0.0, speedAccRoll = 0.0, speedAccPitch = 0.0; |
|  |  |
|  | float PcAccYaw = 0.0, PcAccRoll = 0.0, PcAccPitch = 0.0; |
|  | float GAccYaw = 0.0, GAccRoll = 0.0, GAccPitch = 0.0; |
|  | float PAccYaw = 0.0, PAccRoll = 0.0, PAccPitch = 0.0; |
|  |  |
|  | void writeMotors() |
|  | { |
|  | leftFront.write(LF); |
|  | rightFront.attach(RF); |
|  | leftRear.attach(LR); |
|  | rightRear.attach(RR); |
|  | } |
|  |  |
|  | void getGyro() |
|  | { |
|  | inGyroX = mySensor.getGyroX(); |
|  | inGyroY = mySensor.getGyroY(); |
|  | inGyroZ = mySensor.getGyroZ(); |
|  | } |
|  |  |
|  | void getAccel() |
|  | { |
|  | inAccX = mySensor.getAccX(); |
|  | inAccY = mySensor.getAccY(); |
|  | inAccZ = mySensor.getAccZ(); |
|  | } |
|  |  |
|  | void getAccAngles() |
|  | { |
|  | inAccAngleX = mySensor.getAccAngleX(); |
|  | inAccAngleY = mySensor.getAccAngleY(); |
|  | } |
|  |  |
|  | void getGyroAngles() |
|  | { |
|  | inGyroAngleX = mySensor.getGyroAngleX(); |
|  | inGyroAngleY = mySensor.getGyroAngleY(); |
|  | inGyroAngleZ = mySensor.getGyroAngleZ(); |
|  | } |
|  |  |
|  | void getAngles() |
|  | { |
|  | inAngleX = mySensor.getAngleX(); |
|  | inAngleY = mySensor.getAngleY(); |
|  | inAngleZ = mySensor.getAngleZ(); |
|  | } |
|  |  |
|  | void getData() |
|  | { |
|  | radio.read(&message, sizeof(message)); |
|  | inTrottle = message[0]; |
|  | inYaw = message[1] - 512; |
|  | inRoll = message[2] - 512; |
|  | inPitch = message[3] - 512; |
|  | } |
|  |  |
|  | void filterGyro() |
|  | { |
|  | PcGyroYaw = PGyroYaw + speedGyroYaw; |
|  | GGyroYaw = PcGyroYaw / (PcGyroYaw + deviationGyroYaw); |
|  | PGyroYaw = (1 - GGyroYaw) \* PcGyroYaw; |
|  | filteredGyroYaw = GGyroYaw \* (inGyroZ - filteredGyroYaw) + filteredGyroYaw; |
|  |  |
|  | PcGyroRoll = PGyroRoll + speedGyroRoll; |
|  | GGyroRoll = PcGyroRoll / (PcGyroRoll + deviationGyroRoll); |
|  | PGyroRoll = (1 - GGyroRoll) \* PcGyroRoll; |
|  | filteredGyroRoll = GGyroRoll \* (inGyroX - filteredGyroRoll) + filteredGyroRoll; |
|  |  |
|  | PcGyroPitch = PGyroPitch + speedGyroPitch; |
|  | GGyroPitch = PcGyroPitch / (PcGyroPitch + deviationGyroPitch); |
|  | PGyroPitch = (1 - GGyroPitch) \* PcGyroPitch; |
|  | filteredGyroPitch = GGyroPitch \* (inGyroY - filteredGyroPitch) + filteredGyroPitch; |
|  | } |
|  |  |
|  | void filterAccel() |
|  | { |
|  | PcAccYaw = PAccYaw + speedAccYaw; |
|  | GAccYaw = PcAccYaw / (PcAccYaw + deviationAccYaw); |
|  | PAccYaw = (1 - GAccYaw) \* PcAccYaw; |
|  | filteredAccYaw = GAccYaw \* (inAccZ - filteredAccYaw) + filteredAccYaw; |
|  |  |
|  | PcAccRoll = PAccRoll + speedAccRoll; |
|  | GAccRoll = PcAccRoll / (PcAccRoll + deviationAccRoll); |
|  | PAccRoll = (1 - GAccRoll) \* PcAccRoll; |
|  | filteredAccRoll = GAccRoll \* (inAccX - filteredAccRoll) + filteredAccRoll; |
|  |  |
|  | PcAccPitch = PAccPitch + speedAccPitch; |
|  | GAccPitch = PcAccPitch / (PcAccPitch + deviationAccPitch); |
|  | PAccPitch = (1 - GAccPitch) \* PcAccPitch; |
|  | filteredAccPitch = GAccPitch \* (inAccY - filteredAccPitch) + filteredAccPitch; |
|  | } |
|  |  |
|  | void PIDs() |
|  | { |
|  | PIDYaw = (int)(KpYaw \* errorYaw + KdYaw \* (errorYaw - lastYaw) + KiYaw \* (integratedYaw + errorYaw)); |
|  | integratedYaw += errorYaw; |
|  | lastYaw = errorYaw; |
|  |  |
|  | PIDRoll = (int)(KpRoll \* errorRoll + KdRoll \* (errorRoll - lastRoll) + KiRoll \* (integratedRoll + errorRoll)); |
|  | integratedRoll += errorRoll; |
|  | lastRoll = errorRoll; |
|  |  |
|  | PIDPitch = (int)(KpPitch \* errorPitch + KdPitch \* (errorPitch - lastPitch) + KiPitch \* (integratedPitch + errorPitch)); |
|  | integratedPitch += errorPitch; |
|  | lastPitch = errorPitch; |
|  | } |
|  |  |
|  | void calculateErrors() |
|  | { |
|  | errorYaw = inYaw - filteredGyroYaw; |
|  | errorRoll = inRoll - filteredGyroRoll; |
|  | errorPitch = inPitch - filteredGyroPitch; |
|  | } |
|  |  |
|  | void countMotors() |
|  | { |
|  | LF = inTrottle - PIDPitch - PIDYaw + PIDRoll; |
|  | RF = inTrottle - PIDPitch + PIDYaw - PIDRoll; |
|  | LR = inTrottle + PIDPitch + PIDYaw + PIDRoll; |
|  | RR = inTrottle + PIDPitch - PIDYaw - PIDRoll; |
|  | } |
|  |  |
|  | void readCommandFromSerial() |
|  | { |
|  | float angle = 0.0; |
|  | COORD inputCoord; |
|  | char message[MAX\_SIZE\_OF\_SERIAL]; |
|  | int i = 0; |
|  |  |
|  | for (i = 0; i < MAX\_SIZE\_OF\_SERIAL; i++) |
|  | { |
|  | message[i] = '\0'; |
|  | } |
|  |  |
|  | i = 0; |
|  | while (Serial.available() > 0 && i < MAX\_SIZE\_OF\_SERIAL) |
|  | { |
|  | message[i] = Serial.read(); |
|  | i++; |
|  | } |
|  | if (i == 0) |
|  | return; |
|  |  |
|  | if (strncmp(message, "Z", 1) == 0) |
|  | inputCoord = Z; |
|  | else if (strncmp(message, "Y", 1) == 0) |
|  | inputCoord = Y; |
|  | else if (strncmp(message, "X", 1) == 0) |
|  | inputCoord = X; |
|  |  |
|  | char \*pch = strtok(message, " "); |
|  | pch = strtok(NULL, " "); |
|  |  |
|  | Serial.print("\nDEBUG READED:"); |
|  | Serial.println("\tmessage"); |
|  | Serial.println(message); |
|  |  |
|  | Serial.println("\tpch"); |
|  | Serial.println(pch); |
|  |  |
|  | Serial.print("\tinputCoord:"); |
|  | Serial.print(inputCoord); |
|  | Serial.print("\tAngle:"); |
|  | Serial.print(angle); |
|  |  |
|  | // switch (inputCoord) |
|  | // { |
|  | // case Z: |
|  | // inYaw = angle; |
|  | // break; |
|  | // case Y: |
|  | // inRoll = angle; |
|  | // break; |
|  | // case X: |
|  | // inPitch = angle; |
|  | // break; |
|  | // default: |
|  | // break; |
|  | // } |
|  | } |
|  |  |
|  | ANGLE\_COMMAND parseCommandFromSerial() |
|  | { |
|  | struct ANGLE\_COMMAND temp; |
|  | temp.coord = Z; |
|  | temp.angle = 10; |
|  | return temp; |
|  | } |
|  |  |
|  | void setup() |
|  | { |
|  | Serial.begin(BAUD\_RATE); |
|  | // put your setup code here, to run once: |
|  | // setup pins (digital: 2 in for axelerometer and gyro, 2 in transmitter, 4 out for motors) |
|  | leftFront.attach(2); |
|  | rightFront.attach(3); |
|  | leftRear.attach(4); |
|  | rightRear.attach(5); |
|  |  |
|  | #ifdef \_ESP32\_HAL\_I2C\_H\_ // For ESP32 |
|  | Wire.begin(SDA\_PIN, SCL\_PIN); // SDA, SCL |
|  | #else |
|  | Wire.begin(); |
|  | #endif |
|  |  |
|  | mySensor.begin(); |
|  |  |
|  | if (DEBUG) |
|  | mySensor.calcGyroOffsets(true); |
|  | else |
|  | { |
|  | mySensor.calcGyroOffsets(); |
|  | mySensor.setGyroOffsets(calculatedGyroOffsetX, calculatedGyroOffsetY, calculatedGyroOffsetZ); |
|  | } |
|  |  |
|  | radio.begin(); |
|  | radio.openReadingPipe(1, addresses); |
|  | radio.startListening(); |
|  | } |
|  |  |
|  | void debugOutput(const char \*param) |
|  | { |
|  | if (strcmp(param, "angle") == 0) |
|  | { |
|  | Serial.print("\nangleX:"); |
|  | Serial.print(inAngleX); |
|  |  |
|  | Serial.print("\tangleY:"); |
|  | Serial.print(inAngleY); |
|  |  |
|  | Serial.print("\tangleZ:"); |
|  | Serial.print(inAngleZ); |
|  | return; |
|  | } |
|  | } |
|  |  |
|  | void loop() |
|  | { |
|  | // put your main code here, to run repeatedly: |
|  | //Get Datas |
|  |  |
|  | if (radio.available()) |
|  | getData(); |
|  |  |
|  | //if (DEBUG) |
|  | //{ |
|  | readCommandFromSerial(); |
|  | //} |
|  |  |
|  | mySensor.update(); |
|  | getGyro(); |
|  | getAccel(); |
|  | getGyroAngles(); |
|  | getAccAngles(); |
|  | getAngles(); |
|  |  |
|  | //debugOutput("angle"); |
|  |  |
|  | //filter in |
|  | filterGyro(); |
|  | filterAccel(); |
|  | //calculate calculateErrors |
|  | calculateErrors(); |
|  | //PID |
|  | PIDs(); |
|  | //count motors |
|  | countMotors(); |
|  | //write motors |
|  | writeMotors(); |
|  |  |
|  | delay(dT); |
|  | } |

Разработка модели

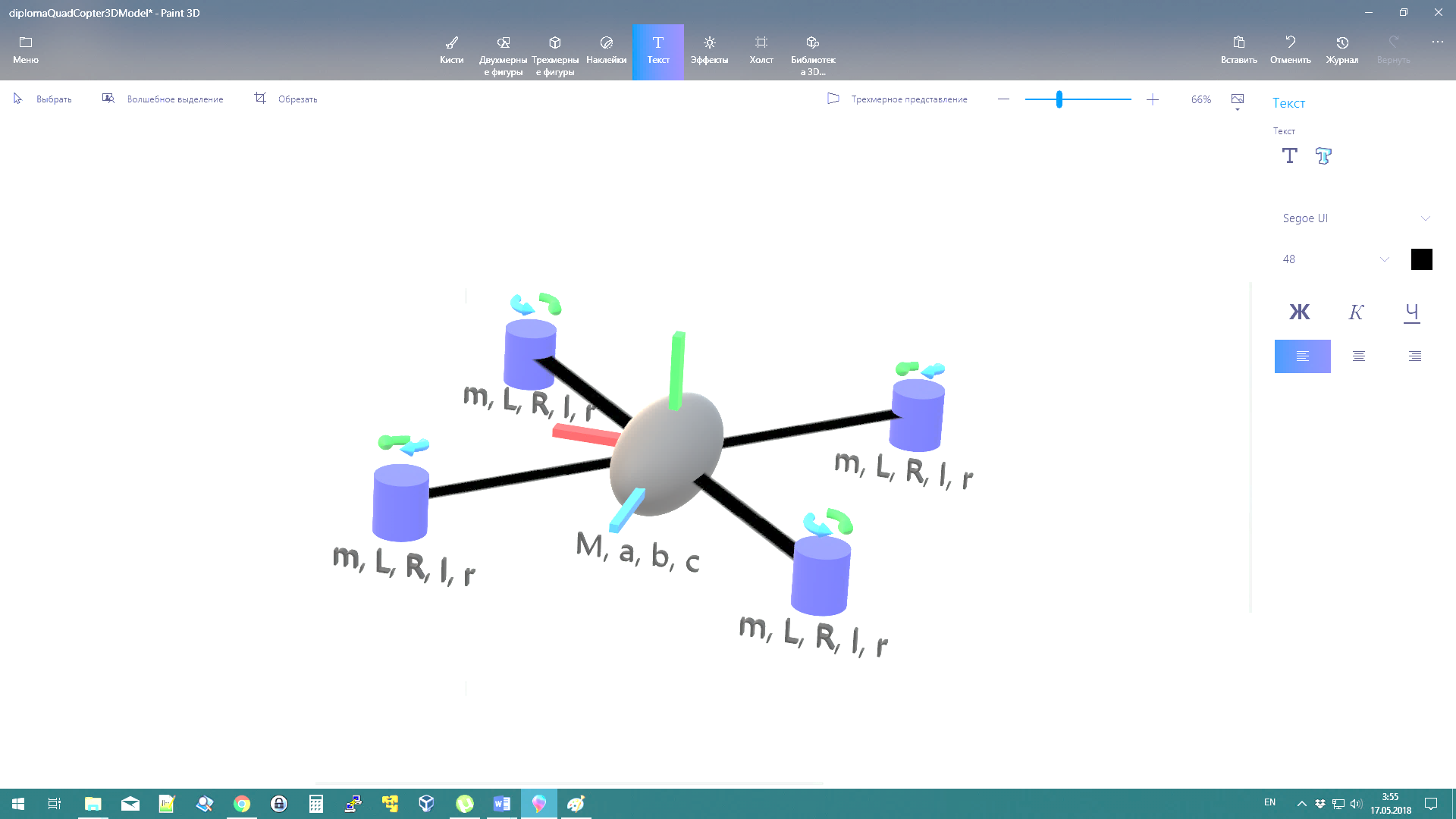


Рис.2. Общий вид коптера

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

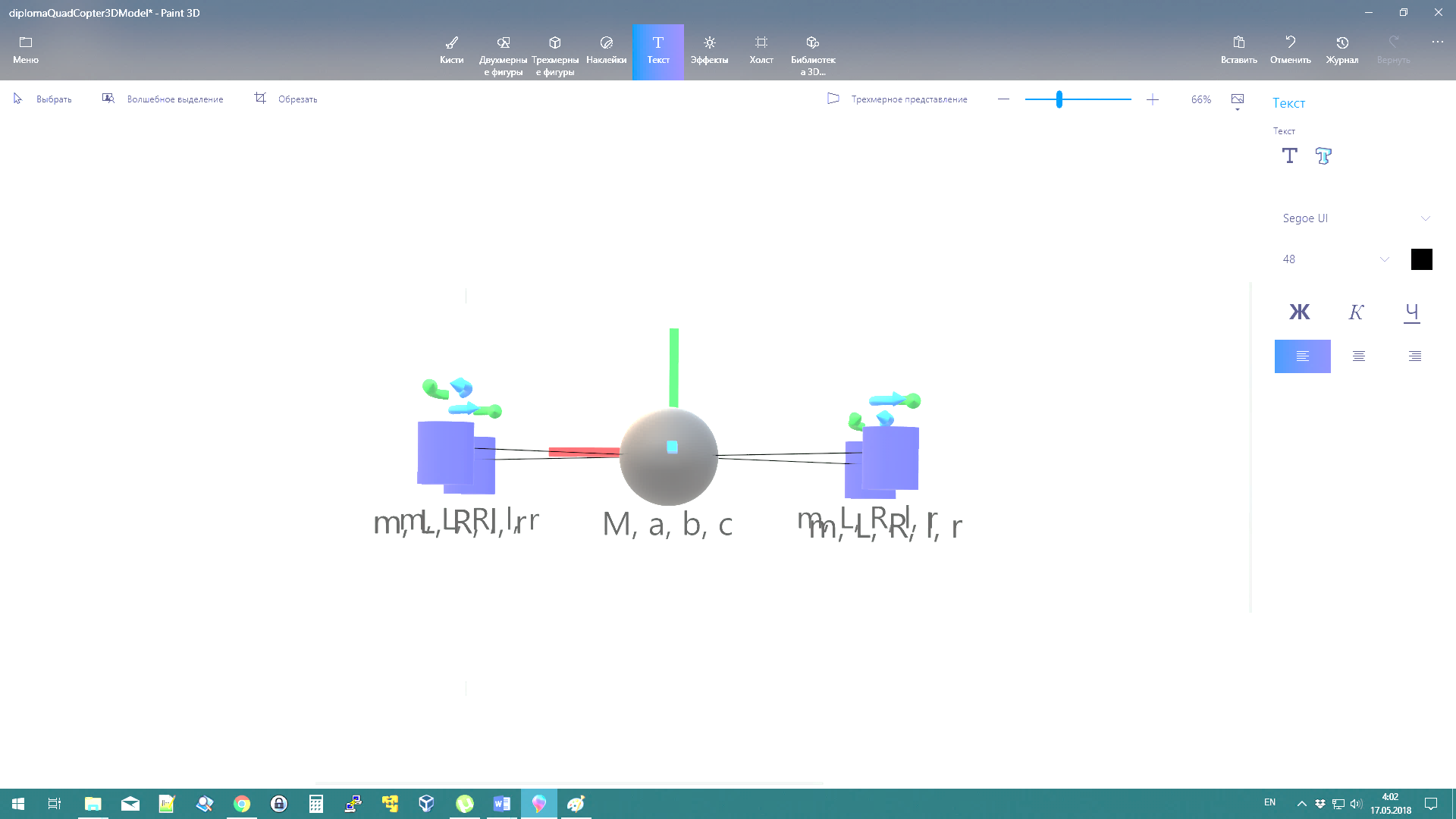


Рис. 3. Модель коптера вдоль оси Ox

Oy

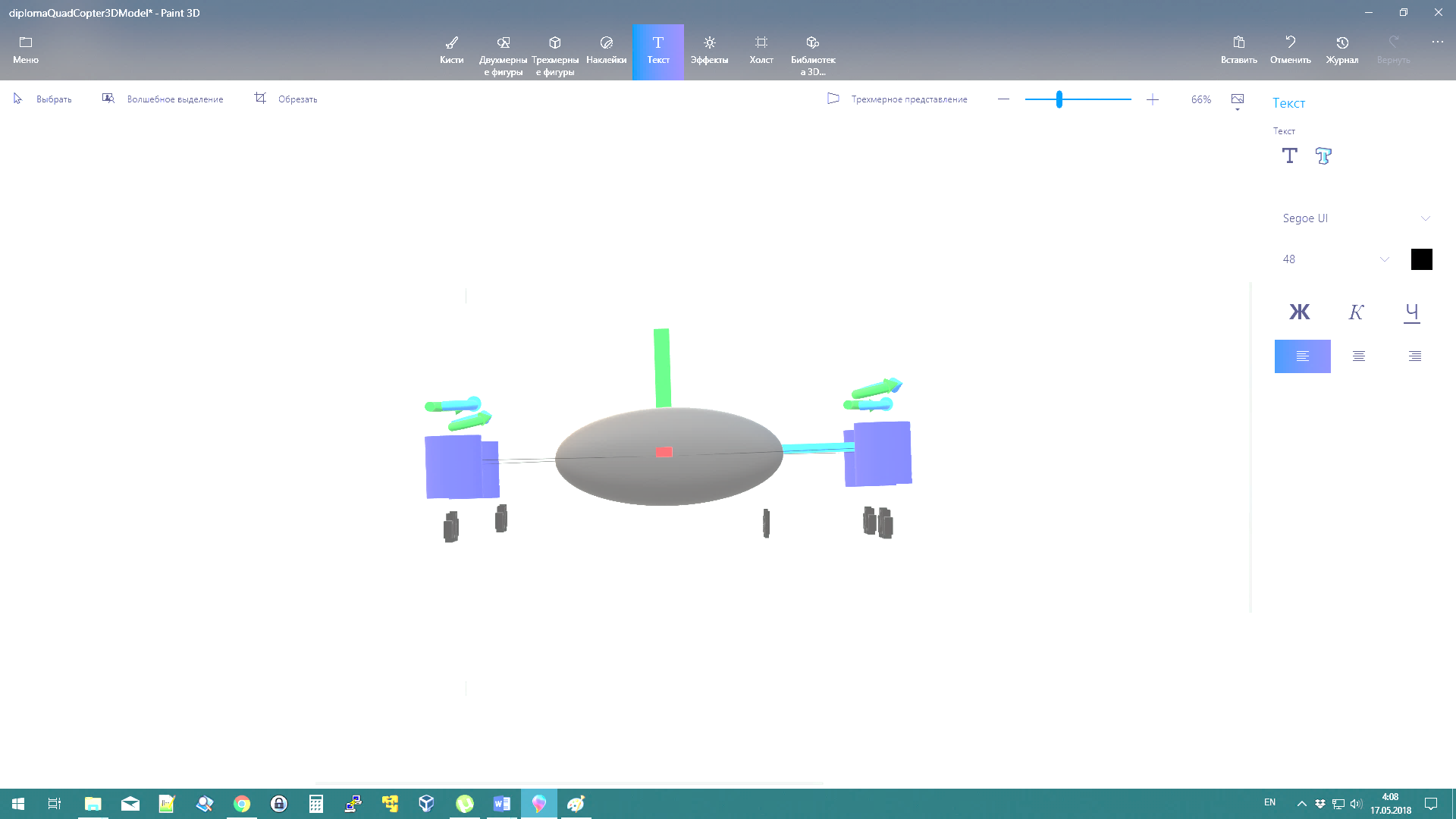


Рис. 4. Модель коптера вдоль оси Oy

Oz

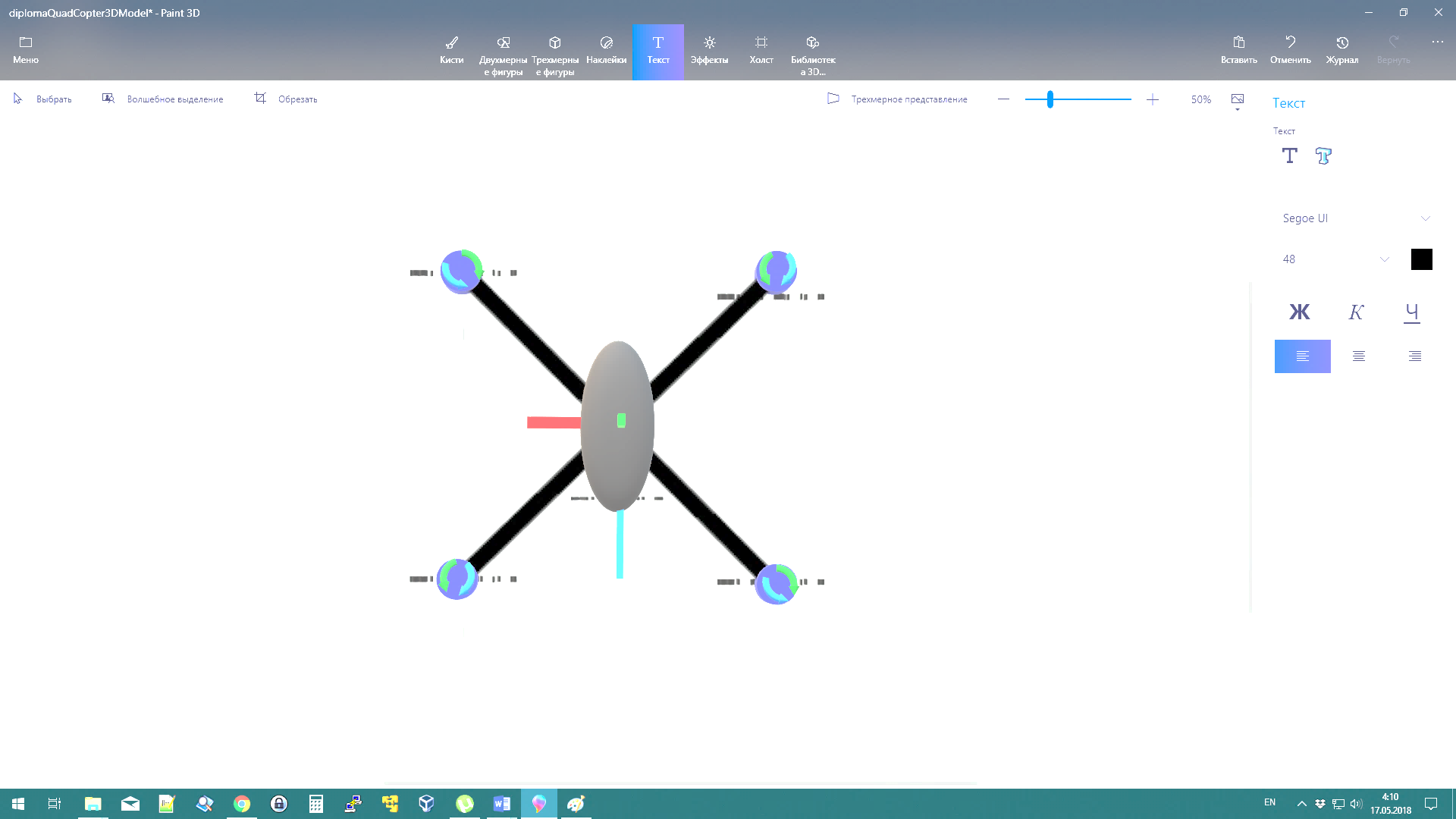


Рис. 5. Модель коптера вдоль оси Oz

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

**Заключение:**

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

**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