ДОДАТОК А

Текст програми

```
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <image_transport/subscriber_filter.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include "std_msgs/String.h"
#include "opencv2/opencv.hpp"
#include <sstream>
#include <message_filters/subscriber.h>
#include <message_filters/time_synchronizer.h>
#include <message_filters/synchronizer.h>
#include <message_filters/sync_policies/approximate_time.h>
#include <geometry_msgs/Twist.h>
#include <geometry_msgs/Pose.h>

using namespace cv;
using namespace std;

typedef message_filters::sync_policies::ApproximateTime<
        sensor_msgs::Image, sensor_msgs::Image
> MySyncPolicy;



//initialise variables which are required for the algorithm
Mat imgPrev, imgLast, imgPrevc, imgLastc, imgPrevNew, imgPrevOld, imgLastOld;
TermCriteria termcrit = TermCriteria(TermCriteria::COUNT + TermCriteria::EPS, 30, 0.01);
vector<float> err;
Size winSize(31, 31);
vector<uchar> statusLKT;
vector<KeyPoint> keypointsPrev, keypointsLast, keypointsPrevNew, goodKeypointsStereoPrev, goodKeypointsStereoLast;
Mat descriptorsPrev, descriptorsLast, descriptorsPrevNew, goodDescriptorsStereo;
OrbDescriptorExtractor extractor(2000);
std::vector< DMatch > matches;
double avrgTime, tick;
std::vector<Point2f>  goodPointsNew, goodPointsTriPrev, goodPointsTriLast;
Mat K1, D1, P1, P2, HandEye, RTnew, Rnew, Rtotal, Ttotal, intrinsics, distortion, rvec, tvec;
```

```
Mat RTtotal = (Mat_<double>(4, 4) << 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1);
Mat tvecSaved = (Mat_<double>(3, 1) << 0, 0, 0);
int initFeatures = 0;
int maxFeat = 0;
double bestAngle = 0;

//function for calculating corresponding feature points
void calcFeatureSets(vector<Point2f>& goodPointsNew, vector<Point2f>& goodPointsTriPrev, vector<Point2f>& goodPointsTriLast)
{
	//clear all old datasets
	goodPointsNew.clear();
	goodPointsTriPrev.clear();
	goodPointsTriLast.clear();
	keypointsPrevNew.clear();
	goodKeypointsStereoPrev.clear();
	goodKeypointsStereoLast.clear();
	keypointsPrev.clear();
	keypointsLast.clear();

	//detect FAST features in prev and last image
	extractor.detect(imgPrevOld, keypointsPrev);
	extractor.detect(imgLastOld, keypointsLast);
	//extract ORB descripors
	extractor.compute(imgPrevOld, keypointsPrev, descriptorsPrev);
	extractor.compute(imgLastOld, keypointsLast, descriptorsLast);
	//cout<<"Features detected: "<<keypointsPrev.size()<<endl;

//Match feature descriptors with brute force matcher and cross check (using hamming distance)
	BFMatcher matcher = BFMatcher(NORM_HAMMING, true);
	matches.clear();
	matcher.match(descriptorsPrev, descriptorsLast, matches);

	/*
	//Ratio Test
	vector<vector< DMatch> > twoMatches;
	matcher.knnMatch(descriptorsPrev,descriptorsLast,twoMatches, 2);
	for (int i=0;i<twoMatches.size();i++){
		if(twoMatches[i][0].distance < 0.9*twoMatches[i][1].distance){
			matches.push_back(twoMatches[i][0]);
			}
	}
		cout<<"Matches Stereo: "<<matches.size()<<endl;
	*/


	//remove bad stereo matches that are not on the same y-coordinate
	std::vector< DMatch > good_matches_stereo, veryGood;
```

```
for (int i = 0; i < matches.size(); i++)
{
	if (fabs(keypointsPrev[matches[i].queryIdx].pt.y - keypointsLast[matches[i].trainIdx].pt.y) <= 2) {
		good_matches_stereo.push_back(matches[i]);
	}
}


//remove matches with a distance higher than a threshol
for (int i = 0; i < good_matches_stereo.size(); i++) {
	if (good_matches_stereo[i].distance < 40) {
		veryGood.push_back(good_matches_stereo[i]);
	}
}

good_matches_stereo = veryGood;
//cout<<"Very Good Matches: "<<veryGood.size()<<endl;

for (int i = 0; i < good_matches_stereo.size(); i++) {
	goodKeypointsStereoPrev.push_back(keypointsPrev[good_matches_stereo[i].queryIdx]);
	goodKeypointsStereoLast.push_back(keypointsLast[good_matches_stereo[i].trainIdx]);
}

vector<Point2f> goodPointsStereoPrev, goodPointsStereoLast, pointsNew;
for (int i = 0; i < goodKeypointsStereoPrev.size(); i++) {
	goodPointsStereoPrev.push_back(goodKeypointsStereoPrev[i].pt);
	goodPointsStereoLast.push_back(goodKeypointsStereoLast[i].pt);
}


//remove outliers with RANSAC
Mat statusStereo;
findFundamentalMat(goodPointsStereoPrev, goodPointsStereoLast, CV_FM_RANSAC, 3, 0.999, statusStereo);

int stereoInliers = 0;
vector <KeyPoint> Prev, Last;
for (int i = 0; i < statusStereo.rows; i++)
{
	if (statusStereo.at<bool>(0, i)) {
		Prev.push_back(goodKeypointsStereoPrev[i]);
		Last.push_back(goodKeypointsStereoLast[i]);
		stereoInliers++;
	}
}
//cout<<"Stereo Inliers: "<<stereoInliers<<endl;
goodKeypointsStereoPrev = Prev;
goodKeypointsStereoLast = Last;
```

```
	goodPointsStereoPrev.clear(); goodPointsStereoLast.clear();
	for (int i = 0; i < goodKeypointsStereoPrev.size(); i++) {
		goodPointsStereoPrev.push_back(goodKeypointsStereoPrev[i].pt);
		goodPointsStereoLast.push_back(goodKeypointsStereoLast[i].pt);
	}


	//track features in current frame based of features in old farme
	calcOpticalFlowPyrLK(imgPrevOld, imgPrev, goodPointsStereoPrev, pointsNew, statusLKT, err, winSize, 3, termcrit, 0, 0.001);

	//generate sets of corresponding keypoints of old stereo images (for triangulation) and new Prev image
	int count = 0;
	for (int i = 0; i < statusLKT.size(); i++)
	{
		if (statusLKT[i]) {
			goodPointsNew.push_back(pointsNew[i]);
			goodPointsTriPrev.push_back(goodKeypointsStereoPrev[i].pt);
			goodPointsTriLast.push_back(goodKeypointsStereoLast[i].pt);
			count++;
		}
	}
	cout << "matched features: " << count << endl;
}



//triangulate 3D points with the linear method
Mat Triangulation(Point2f u, Mat P, Point2f u1, Mat P1)
{
	Mat A = (Mat_<double>(4, 3) << u.x * P.at<double>(2, 0) - P.at<double>(0, 0), u.x * P.at<double>(2, 1) - P.at<double>(0, 1), u.x * P.at<double>(2, 2) - P.at<double>(0, 2),
		u.y * P.at<double>(2, 0) - P.at<double>(1, 0), u.y * P.at<double>(2, 1) - P.at<double>(1, 1), u.y * P.at<double>(2, 2) - P.at<double>(1, 2),
		u1.x * P1.at<double>(2, 0) - P1.at<double>(0, 0), u1.x * P1.at<double>(2, 1) - P1.at<double>(0, 1), u1.x * P1.at<double>(2, 2) - P1.at<double>(0, 2),
		u1.y * P1.at<double>(2, 0) - P1.at<double>(1, 0), u1.y * P1.at<double>(2, 1) - P1.at<double>(1, 1), u1.y * P1.at<double>(2, 2) - P1.at<double>(1, 2)
		);
	Mat B = (Mat_<double>(4, 1) << -(u.x * P.at<double>(2, 3) - P.at<double>(0, 3)),
		-(u.y * P.at<double>(2, 3) - P.at<double>(1, 3)),
		-(u1.x * P1.at<double>(2, 3) - P1.at<double>(0, 3)),
		-(u1.y * P1.at<double>(2, 3) - P1.at<double>(1, 3)));

	Mat X;
	solve(A, B, X, DECOMP_SVD);

	return X;
```

```
}


//ROS class which receives the rectified images and publishes the pose of the robot
class ImageConverter
{
public:
	ImageConverter();
	void InitializePubSub();
private:
	ros::NodeHandle nh_;
	image_transport::ImageTransport it_;
	//image subscriber
	image_transport::SubscriberFilter Prev_image_sub_;
	image_transport::SubscriberFilter Last_image_sub_;
	message_filters::Synchronizer< MySyncPolicy > sync_;
	//pose Publisher
	ros::Publisher chatter_pub = nh_.advertise<geometry_msgs::Pose>("/cmd_pos", 100);
	//Subscriber Callback method
	void stereoCallback(
		const sensor_msgs::ImageConstPtr& Prev_image_msg,
		const sensor_msgs::ImageConstPtr& Last_image_msg);
};

void ImageConverter::InitializePubSub() {
	sync_.registerCallback(boost::bind(&ImageConverter::stereoCallback, this, _1, _2));
}

//for implementation on the apollon notebook, subscribe the following: Prev_image_sub_(it_, "/apollon/vrmagic/Prev/image_rect_color", 1),
Last_image_sub_(it_, "/apollon/vrmagic/Last/image_rect_color", 1)
ImageConverter::ImageConverter() :
	it_(nh_), Prev_image_sub_(it_, "/vrmagic/Prev/image_rect_color", 1), Last_image_sub_(it_, "/vrmagic/Last/image_rect_color", 1),
	sync_(MySyncPolicy(10), Prev_image_sub_, Last_image_sub_) {
	InitializePubSub();
}

//converting ROS image Message to OpenCV Mat with CVBridge
void ImageConverter::stereoCallback(const sensor_msgs::ImageConstPtr& msg1, const sensor_msgs::ImageConstPtr& msg2)
{
	cv_bridge::CvImagePtr cv_ptr1, cv_ptr2;
	try
	{
		cv_ptr1 = cv_bridge::toCvCopy(msg1, "bgr8");
		cv_ptr2 = cv_bridge::toCvCopy(msg2, "bgr8");
	}
	catch (cv_bridge::Exception& e)
	{
```

```
		ROS_ERROR("cv_bridge exception: %s", e.what());
		return;
}

//VO ALGORITHM:
//start clock and load images
tick = (double)getTickCount();
imgPrevc = cv_ptr1->image;
imgLastc = cv_ptr2->image;

//convert images to grayscale
cvtColor(imgPrevc, imgPrev, COLOR_BGR2GRAY);
cvtColor(imgLastc, imgLast, COLOR_BGR2GRAY);

if (imgPrevOld.empty()) {
		imgPrev.copyTo(imgPrevOld);
		imgLast.copyTo(imgLastOld);
}

//detect and match features in the previous Prev and Last image (for triangulation) and track those features in the new Prev image
calcFeatureSets(goodPointsNew, goodPointsTriPrev, goodPointsTriLast);

//triangulate goodPointsTriPrev and goodPointsTriLast to get a set of 3D worldPoints
if (goodPointsTriPrev.size() > 5) {
		std::vector<Point3f>  worldPointsHart;
		for (int i = 0; i < goodPointsTriPrev.size(); i++) {
			Mat world = Triangulation(goodPointsTriPrev.at(i), P1, goodPointsTriLast.at(i), P2);
			worldPointsHart.push_back(Point3f(world.at<double>(0, 0), world.at<double>(1, 0), world.at<double>(2, 0)));
		}

		//Solve PnP with RANSAC
		Mat inliers;
		solvePnPRansac(worldPointsHart, goodPointsNew, K1, D1, rvec, tvec, false, 1000, 2.0, -1, inliers, CV_P3P);
		tvec = -tvec;
		rvec = -rvec;
		//print out current motion
		cout << "R: " << rvec * 180 / 3.14 << endl;
		cout << "T: " << tvec << endl;

		/*
		//calculate the inliers of the PnP algorithm
		int inlier=0;
		for( int i = 0; i < inliers.rows; i++ )
		 {
			if(inliers.at<bool>(0,i)){
				inlier++;
			}
```

```
 }
cout<<"PnP Inliers: "<<inlier<<endl;
*/

//remove very small or large rotations and translations
rvec.at<double>(0) = 0;
rvec.at<double>(2) = 0;
tvec.at<double>(1) = 0;
for (int i = 0; i < 3; i++) {
	if ((fabs(rvec.at<double>(i)) < 0.2 / 180 * 3.14) || (fabs(rvec.at<double>(i)) > 1.5)) {
		rvec.at<double>(i) = 0;
	}
}
for (int i = 0; i < 3; i++) {
	if ((fabs(tvec.at<double>(i)) < 1) || (fabs(tvec.at<double>(i)) > 300)) {
		tvec.at<double>(i) = 0;
	}
}


//Calculate total pose
Rodrigues(rvec, Rnew);

RTnew = (Mat_<double>(4, 4) << Rnew.at<double>(0, 0), Rnew.at<double>(0, 1), Rnew.at<double>(0, 2),
tvec.at<double>(0), Rnew.at<double>(1, 0), Rnew.at<double>(1, 1), Rnew.at<double>(1, 2), tvec.at<double>(1), Rnew.at<double>(2, 0),
Rnew.at<double>(2, 1), Rnew.at<double>(2, 2), tvec.at<double>(2), 0, 0, 0, 1);

//Calculating the robots pose from the cameras pose (Hand-Eye-Relation)
RTnew = HandEye * RTnew * HandEye.inv();
RTtotal = RTtotal * RTnew;

//print out total pose
Rtotal = (Mat_<double>(3, 3) << RTtotal.at<double>(0, 0), RTtotal.at<double>(0, 1), RTtotal.at<double>(0, 2),
RTtotal.at<double>(1, 0), RTtotal.at<double>(1, 1), RTtotal.at<double>(1, 2), RTtotal.at<double>(2, 0), RTtotal.at<double>(2, 1),
RTtotal.at<double>(2, 2));
Ttotal = (Mat_<double>(3, 1) << RTtotal.at<double>(0, 3), RTtotal.at<double>(1, 3), RTtotal.at<double>(2, 3));
Rodrigues(Rtotal, rvec);

cout << "R Total: " << rvec * 180 / 3.14 << endl;
cout << "T Total: " << Ttotal << endl;


//VISUALISATION:
//draw lines for moving Keypoints that moved more than a threshold
for (int i = 0; i < goodPointsNew.size(); i++) {
	Point p0(ceil(goodPointsNew[i].x), ceil(goodPointsNew[i].y));
	Point p1(ceil(goodPointsTriPrev[i].x), ceil(goodPointsTriPrev[i].y));
```

```
                double res = cv::norm(p1 - p0);
                if (res > 5) {
                        line(imgPrevc, p0, p1, CV_RGB(0, 255, 0), 2);
                }
        }

        //calculate colour according to depth of the 3D point
        vector<float> depth;
        float maxDepth = 0;
        float minDepth = 10000;
        for (int i = 0; i < worldPointsHart.size(); i++) {
                if (worldPointsHart.at(i).z > 0) {
                        if (worldPointsHart.at(i).z > maxDepth) {
                                maxDepth = worldPointsHart.at(i).z;
                        }
                        if (worldPointsHart.at(i).z < minDepth) {
                                minDepth = worldPointsHart.at(i).z;
                        }
                }
        }
        for (int i = 0; i < goodPointsTriPrev.size(); i++) {
                depth.push_back((worldPointsHart.at(i).z - minDepth) / (maxDepth - minDepth) * 255);
        }

        //draw feature points and depth information
        for (int i = 0; i < goodPointsTriPrev.size(); i++) {
                Point p0(ceil(goodPointsTriPrev[i].x), ceil(goodPointsTriPrev[i].y));
                Point p1(ceil(goodPointsTriLast[i].x), ceil(goodPointsTriLast[i].y));
                Scalar colorDepth = Scalar(0, depth[i], 255);
                circle(imgPrevc, p0, 2, Scalar(0, 255, 0), 2);
                circle(imgLastc, p1, 2, colorDepth, 2);
        }
}


//Output Images and copy current images to old images
imshow("Input1", imgPrevc);
imshow("Input2", imgLastc);

//save the current image frame for the next calculation step
imgPrev.copyTo(imgPrevOld);
imgLast.copyTo(imgLastOld);

//publish current pose
geometry_msgs::Pose cmd_pos_msg;
cmd_pos_msg.position.x = Ttotal.at<double>(0, 0);
```

```
cmd_pos_msg.position.y = 0;
cmd_pos_msg.position.z = Ttotal.at<double>(2, 0);
cmd_pos_msg.orientation.x = rvec.at<double>(0, 0);
cmd_pos_msg.orientation.y = rvec.at<double>(1, 0);
cmd_pos_msg.orientation.z = rvec.at<double>(2, 0);


//STATIC FEATURE SEARCH
//initiate feature search if there are too less features
if ((initFeatures == 3) && (goodPointsNew.size() < 30)) {
	tvecSaved = Ttotal;
	initFeatures = 0;
	maxFeat = 0;
}

//turn to the Prev, searching for most features
if (initFeatures == 0) {
	cmd_pos_msg.position.y = 2;
	if (maxFeat < goodPointsNew.size()) {
		maxFeat = goodPointsNew.size();
		bestAngle = rvec.at<double>(1);
	}
	if ((rvec.at<double>(1) < -1.5) || (goodPointsNew.size() < 60)) {
		initFeatures = 1;
	}
}

//turn to the Last, searching for most features
if (initFeatures == 1) {
	cmd_pos_msg.position.y = 1;
	if (maxFeat < goodPointsNew.size()) {
		maxFeat = goodPointsNew.size();
		bestAngle = rvec.at<double>(1);
	}
	if ((rvec.at<double>(1) > 1.5) || ((goodPointsNew.size() < 60) && (rvec.at<double>(1) > 0))) {
		initFeatures = 2;
		cout << "Feature search done! Best angle: " << bestAngle * 180 / 3.14 << " with " << maxFeat << " features." <<
endl;
		waitKey(2000);
	}
}

//turn to the direction with the most features
if (initFeatures == 2) {
	cmd_pos_msg.position.y = 2;
	if (rvec.at<double>(1) <= bestAngle) {
		initFeatures = 3;
```

```
                cmd_pos_msg.position.y = 0;
                RTtotal.at<double>(0, 3) = tvecSaved.at<double>(0);
                RTtotal.at<double>(1, 3) = tvecSaved.at<double>(1);
                RTtotal.at<double>(2, 3) = tvecSaved.at<double>(2);
                cout << "Best angle reached, starting to move!" << endl;
                waitKey(2000);
        }
    }


    /*
        //DYNAMIC FEATURE SEARCH
        if(goodPointsNew.size()<100){
            cmd_pos_msg.position.y=3;
        }
        else {
            cmd_pos_msg.position.y=0;
        }
    */

    //wait 1ms
    char k = waitKey(1);
    if (k == 'a') { k = waitKey(20000); }

    chatter_pub.publish(cmd_pos_msg);

    //calculate time
    avrgTime = ((double)getTickCount() - tick) / getTickFrequency();
    cout << "Time: " << avrgTime * 1000 << "ms" << endl;
    cout << "_____" << endl;
}


    #include <ros/ros.h>
    #include <image_transport/image_transport.h>
    #include <image_transport/subscriber_filter.h>
    #include <cv_bridge/cv_bridge.h>
    #include <sensor_msgs/image_encodings.h>
    #include "std_msgs/String.h"
    #include "opencv2/opencv.hpp"
    #include <sstream>
    #include <message_filters/subscriber.h>
    #include <message_filters/time_synchronizer.h>
    #include <message_filters/synchronizer.h>
    #include <message_filters/sync_policies/approximate_time.h>
    #include <geometry_msgs/Twist.h>
    #include <geometry_msgs/Pose.h>
```

```
using namespace cv;
using namespace std;

//destination trajectory in mm
double destinationsX [4] ={0,1000,1000,0};
double destinationsY [4] ={1000,1000,0,0};
//maximum speed of the robotino
double SPEED=0.2,speed;

double xr,yr,angle,xw,yw;
int k=0;


class Controller
{
        public:
        Controller()
        {
                //initialise pose subscriber and velocity publisher (for robotino_node)
                 chatter_pub=n.advertise<geometry_msgs::Twist>("/cmd_vel", 100);
                 sub = n.subscribe("/cmd_pos", 1000, &Controller::chatterCallback,this);
        }

        void chatterCallback(const geometry_msgs::Pose::ConstPtr& msg)
        {
                //update current destination
                xw=destinationsX[k];
                yw=destinationsY[k];

                //rotate to the right if desired
    if(msg->position.y==1){
                        geometry_msgs::Twist cmd_vel_msg;
                        cmd_vel_msg.angular.z=-0.2;
                        chatter_pub.publish(cmd_vel_msg);
                }
                //rotate to the left if desired
    else if (msg->position.y==2){
       geometry_msgs::Twist cmd_vel_msg;
       cmd_vel_msg.angular.z=0.2;
       chatter_pub.publish(cmd_vel_msg);
    }
                //otherwise move to the destination
                else {
                        //load current pose of the robot
                        xr=msg->position.x;
                        yr=msg->position.z;
```

```
angle=-msg->orientation.y;

                        //calculate distance from robot to destination
                        double dist=fabs(xw-xr)+fabs(yw-yr);

                        //introduce some variables
double xwn,ywn,xvel,yvel,xvelout,yvelout;
                        geometry_msgs::Twist cmd_vel_msg;

                        //rotate the world coorinate system to the robots coordiante system
                        xwn=cos(angle)*(xw-xr)+sin(angle)*(yw-yr);
                        ywn=-sin(angle)*(xw-xr)+cos(angle)*(yw-yr);

                        //adapt coordinate systems
                        xvel=ywn;
                        yvel=-xwn;


cmd_vel_msg.angular.z=0;
                        //check if the destination is close or already reached
if ((dist<=50)&&(k==3)) {
                                speed=0;
                        }
                        //move to the next point of the given trajectory
else if ((dist>50)&&(dist<150)) {
   speed=0.10;
                                if(k!=3){
      k++;
                                }
                        }
                        else {
                                speed=SPEED;
                        }

                        //scale the velocity to maximum
                        if (fabs(xvel)>=fabs(yvel)){
                        xvelout=xvel*speed/fabs(xvel);
                        yvelout=yvel*speed/fabs(xvel);
                        }
                        else {
                        xvelout=xvel*speed/fabs(yvel);
                        yvelout=yvel*speed/fabs(yvel);
                        }

                        //rotate while moving for dynamic feature search
                        if(msg->position.y==3){
                        cmd_vel_msg.angular.z=-0.2;
```

```
                        }

                        //output information
                        cout<<"x velocity: "<<xvelout<<endl;
                        cout<<"y velocity: "<<yvelout<<endl;
                        cout<<"x POS: "<<xr<<endl;
                        cout<<"y POS: "<<yr<<endl;
                        cout<<"Theta: "<<angle*180/M_PI<<endl;
                        cout<<"Distance: "<<dist<<endl;
                        cout<<"_____"<<endl;

                        //publish velocity commands
                        cmd_vel_msg.linear.x=xvelout;
                        cmd_vel_msg.linear.y=yvelout;

                        chatter_pub.publish(cmd_vel_msg);
                }
        }

        private:
          ros::NodeHandle n;
          ros::Publisher chatter_pub;
          ros::Subscriber sub;
        };


#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <image_transport/subscriber_filter.h>
#include <cv_bridge/cv_bridge.h>
#include <message_filters/subscriber.h>

class RobotMoveExecutor
{
public:

Controller(OmniDrive* omniDrive)
{
        _omniDrive = omniDrive;
        sub = n.subscribe(1000, &Controller::executeCallback,this);
}

void executeCallback(const geometry_msgs::Twist& msg)
{
        OmniDrive_setVelocity(_omniDrive, msg.linear.x, msg.linear.y, msg.linear.velAngle);
}
```

```
private:
ros::Subscriber sub;
OmniDrive* _omniDrive = nullptr;
};
```

```
cmake_minimum_required(VERSION 2.8.3)
project(ImageConverter)

find_package(catkin REQUIRED COMPONENTS
  cv_bridge
  image_transport
  roscpp
  sensor_msgs
  std_msgs
  geometry_msgs
)
 find_package(OpenCV REQUIRED)

catkin_package()


include_directories(
  ${catkin_INCLUDE_DIRS}
)
include_directories(${OpenCV_INCLUDE_DIRS})

add_executable(ImageConverter src/main.cpp)
target_link_libraries(localisation ${catkin_LIBRARIES} )
 target_link_libraries(localisation ${OpenCV_LIBRARIES})


cmake_minimum_required(VERSION 2.8.3)
project(Controller)

find_package(catkin REQUIRED COMPONENTS
  cv_bridge
  image_transport
  roscpp
  sensor_msgs
  std_msgs
  geometry_msgs
)
 find_package(OpenCV REQUIRED)

catkin_package()
```

```
include_directories(
  ${catkin_INCLUDE_DIRS}
)
include_directories(${OpenCV_INCLUDE_DIRS})

add_executable(Controller src/main.cpp)
target_link_libraries(control ${catkin_LIBRARIES} )
 target_link_libraries(control ${OpenCV_LIBRARIES})



cmake_minimum_required(VERSION 2.8.3)
project(RobotMoveExecutor)

find_package(catkin REQUIRED COMPONENTS
  cv_bridge
  image_transport
  roscpp
  std_msgs
)

catkin_package()

include_directories(
  ${catkin_INCLUDE_DIRS}
)
include_directories(${OpenCV_INCLUDE_DIRS})

add_executable(RobotMoveExecutor src/main.cpp)
target_link_libraries(control ${catkin_LIBRARIES})



//  Copyright (C) 2004-2008, Robotics Equipment Corporation GmbH

#define _USE_MATH_DEFINES
#include <cmath>
#include <iostream>
#include <stdlib.h>

#ifdef WIN32
#include <windows.h>
#else
#include <signal.h>
#endif

#include <sstream>
#include <stdio.h>
```

```
#include <stdlib.h>

#include <rec/robotino/api2/all.h>

using namespace rec::robotino::api2;

bool _run = true;

enum
{
	PPM_BINARY_IMAGE_OUTPUT,
	PPM_PLAIN_IMAGE_OUTPUT,
	JPG_IMAGE_OUTPUT
};

int imageOutputFormat = PPM_BINARY_IMAGE_OUTPUT;


#ifdef WIN32
static BOOL WINAPI sigint_handler( DWORD fdwCtrlType )
{
	switch( fdwCtrlType )
	{
	case CTRL_C_EVENT: // Handle the CTRL-C signal.
		_run = false;
		return TRUE;

	default:
		return FALSE;
	}
}
#else
void sigint_handler( int signum )
{
	_run = false;
}
#endif

class MyCom : public Com
{
public:
	MyCom()
		: Com( "example_camera" )
	{
	}

	void errorEvent( const char* errorString )
```

```
        {
                std::cerr << "Error: " << errorString << std::endl;
        }

        void connectedEvent()
        {
                std::cout << "Connected." << std::endl;
        }

        void connectionClosedEvent()
        {
                std::cout << "Connection closed." << std::endl;
        }

        void logEvent( const char* message, int level )
        {
                std::cout << message << std::endl;
        }

        void pingEvent( float timeMs )
        {
                std::cout << "Ping: " << timeMs << "ms" << std::endl;
        }
};

class MyCamera : public Camera
{
public:
        MyCamera(OmniDrive* omniDrive) : _omniDrive(omniDrive)
        {
        }

        void imageReceivedEvent( const unsigned char* data,
                                                                                unsigned int
dataSize,
                                                                                unsigned int
width,
                                                                                unsigned int
height,
                                                                                unsigned int step )
        {
                static unsigned int seq = 0;

                int currentImageOutputFormat = imageOutputFormat;
                if (0 == width)
                {
                        currentImageOutputFormat = JPG_IMAGE_OUTPUT;
```

```
}
else
{
	switch (imageOutputFormat)
	{
	case PPM_BINARY_IMAGE_OUTPUT:
	case PPM_PLAIN_IMAGE_OUTPUT:
		currentImageOutputFormat = imageOutputFormat;
		break;

	default:
		currentImageOutputFormat = PPM_PLAIN_IMAGE_OUTPUT;
		break;
	}
}

std::ostringstream os;
os << "image" << seq;

FILE* fp = NULL;

switch (currentImageOutputFormat)
{
case PPM_BINARY_IMAGE_OUTPUT:
case PPM_PLAIN_IMAGE_OUTPUT:
	os << ".ppm";
	fp = fopen(os.str().c_str(), "w");
	break;

default:
	os << ".jpg";
	fp = fopen(os.str().c_str(), "wb");
	break;
}


if ( fp == NULL )
{
	std::cerr << "Error: Cannot open file " << os.str() << std::endl;
	return;
}

std::cout << "Writing " << os.str() << std::endl;

switch (currentImageOutputFormat)
{
case PPM_BINARY_IMAGE_OUTPUT:
```

```
                        fprintf(fp, "P6 %d %d 255\n", width, height);
                        fclose(fp);
                        fp = fopen(os.str().c_str(), "ab");
                        for (unsigned int line = 0; line<height; ++line)
                        {
                                const unsigned char* psrc = (const unsigned char*)data + step * line;
                                fwrite( (const char*)psrc, width * 3, 1, fp);
                        }
                        break;

                case PPM_PLAIN_IMAGE_OUTPUT:
                        fprintf(fp, "P3 %d %d 255\n", width, height);

                        for (unsigned int line = 0; line<height; ++line)
                        {
                                const unsigned char* psrc = (const unsigned char*)data + step * line;
                                for (unsigned int i = 0; i < width * 3; i += 3)
                                {
                                        fprintf(fp, " %d %d %d ", (int)(*(psrc + i)), (int)(*(psrc + i + 1)), (int)(*(psrc + i + 2)));
                                }
                                fprintf(fp, "\n");
                        }
                        break;

                default:
                        fwrite(data, dataSize, 1, fp);
                        break;
                }

                fclose(fp);

                ++seq;
        }

        OmniDrive* _omniDrive = nullptr;
};


MyCom com(OmniDrive_construct());
MyCamera camera;

void init( const std::string& hostname )
{
        // Initialize the actors

        if (JPG_IMAGE_OUTPUT == imageOutputFormat)
        {
```

```
		camera.setJPGDecodingEnabled(false);
	}

	// Connect
	std::cout << "Connecting...";
	com.setAddress( hostname.c_str() );

	com.connectToServer( true );

	if( !com.isConnected() )
	{
		std::cout << std::endl << "Could not connect to " << com.address() << std::endl;
#ifdef WIN32
		std::cout << "Press any key to exit..." << std::endl;
		rec::robotino::api2::waitForKey();
#endif
		rec::robotino::api2::shutdown();
		exit( 1 );
	}
	else
	{
		std::cout << "success" << std::endl;
	}
}

void drive()
{
	while( com.isConnected() )
	{
		com.processEvents();
		ros::init(argc, argv, "localisation");

		ImageConverter ic;
		Controller controller;
		RobotMoveExecutor(com._omniDrive);

		ros::spin();

		rec::robotino::api2::msleep( 1000 );
	}
}

void destroy()
{
	com.disconnectFromServer();
}
```

```
void printHelp()
{
	std::cout << "options:" << std::endl;
	std::cout << "--hostname=ipaddress     : set host to connect to" << std::endl;
	std::cout << "--ppm-binary             : output PPM binary images (default)" << std::endl;
	std::cout << "--ppm-plain              : output PPM plain images" << std::endl;
	std::cout << "--jpg                    : output jpg images (if available)" << std::endl;
	std::cout << "-help | --help | -h | /? : print this help page" << std::endl;
}

int main( int argc, char **argv )
{
	std::string hostname = "127.0.0.1";

	for (int i = 1; i<argc; ++i)
	{
		std::string arg = argv[i];

		if ("--hostname" == arg.substr(0, 10))
		{
			hostname = arg.substr(11,std::string::npos);
		}
		else if ("--ppm-binary" == arg.substr(0, 12))
		{
			imageOutputFormat = PPM_BINARY_IMAGE_OUTPUT;
		}
		else if ("--ppm-plain" == arg.substr(0, 11))
		{
			imageOutputFormat = PPM_PLAIN_IMAGE_OUTPUT;
		}
		else if( "--jpg" == arg.substr(0, 5) )
		{
			imageOutputFormat = JPG_IMAGE_OUTPUT;
		}
		else
		{
			printHelp();
			exit(0);
		}
	}

	switch (imageOutputFormat)
	{
	case PPM_BINARY_IMAGE_OUTPUT:
		std::cout << "PPM binary output enabled" << std::endl;
		break;
```

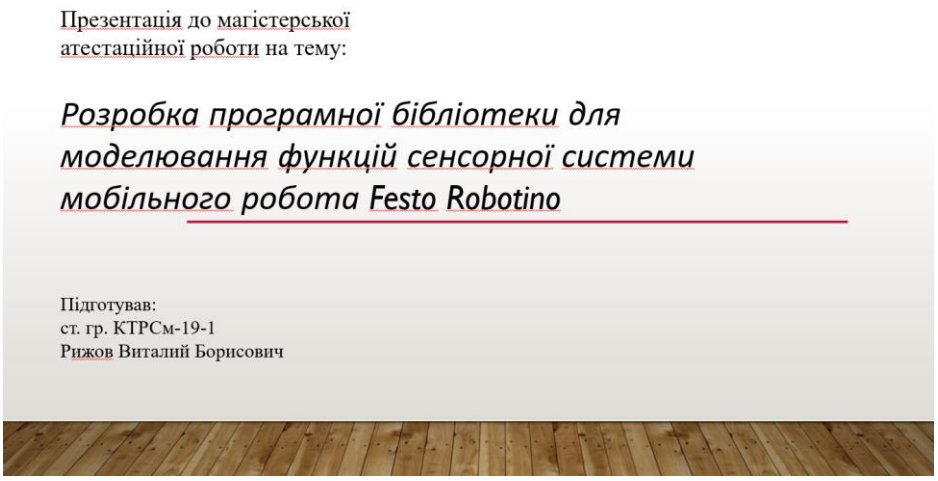
```
		case PPM_PLAIN_IMAGE_OUTPUT:
			std::cout << "PPM plain output enabled" << std::endl;
			break;

		default:
			std::cout << "JPG output enabled" << std::endl;
			break;
		}

#ifdef WIN32
		::SetConsoleCtrlHandler( (PHANDLER_ROUTINE) sigint_handler, TRUE );
#else
		struct sigaction act;
		memset( &act, 0, sizeof( act ) );
		act.sa_handler = sigint_handler;
		sigaction( SIGINT, &act, NULL );
#endif
		try
		{
			init( hostname );
			drive();
			destroy();
		}
		catch( const rec::robotino::api2::RobotinoException& e )
		{
			std::cerr << "Com Error: " << e.what() << std::endl;
		}
		catch( const std::exception& e )
		{
			std::cerr << "Error: " << e.what() << std::endl;
		}
		catch( ... )
		{
			std::cerr << "Unknow Error" << std::endl;
		}

		rec::robotino::api2::shutdown();

#ifdef WIN32
		std::cout << "Press any key to exit..." << std::endl;
		rec::robotino::api2::waitForKey();
#endif
}
```

ДОДАТОК Б

Демонстраційний матеріал

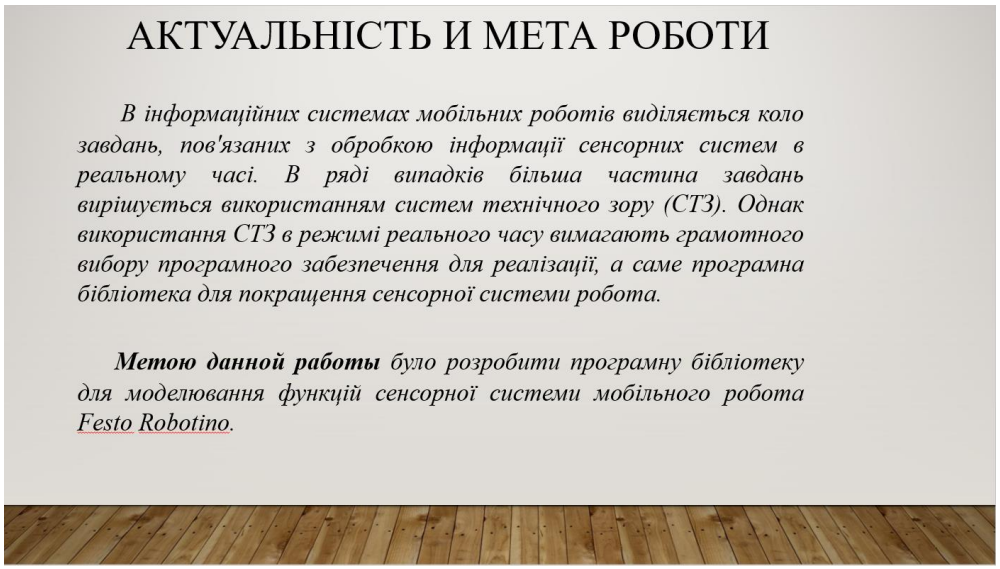
Слайд 1

ТИТУЛЬНИЙ ЛИСТ РОБОТИ



Слайд 2

АКТУАЛЬНІСТЬ ТА МЕТА РОБОТИ

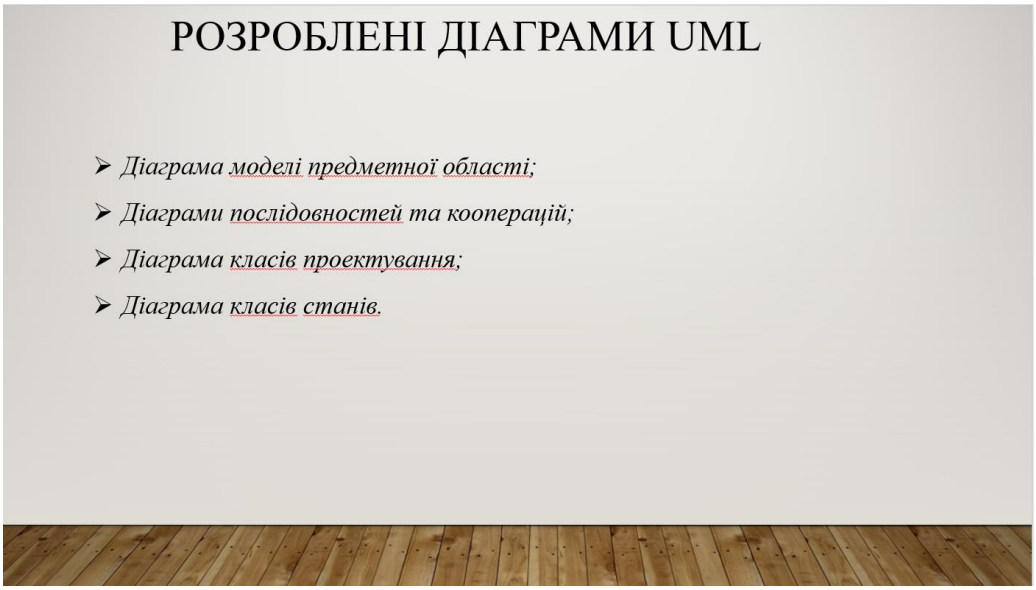


Слайд 3

АРХІТЕКТУРА ПРОГРАМНОЇ БІБЛІОТЕКИ

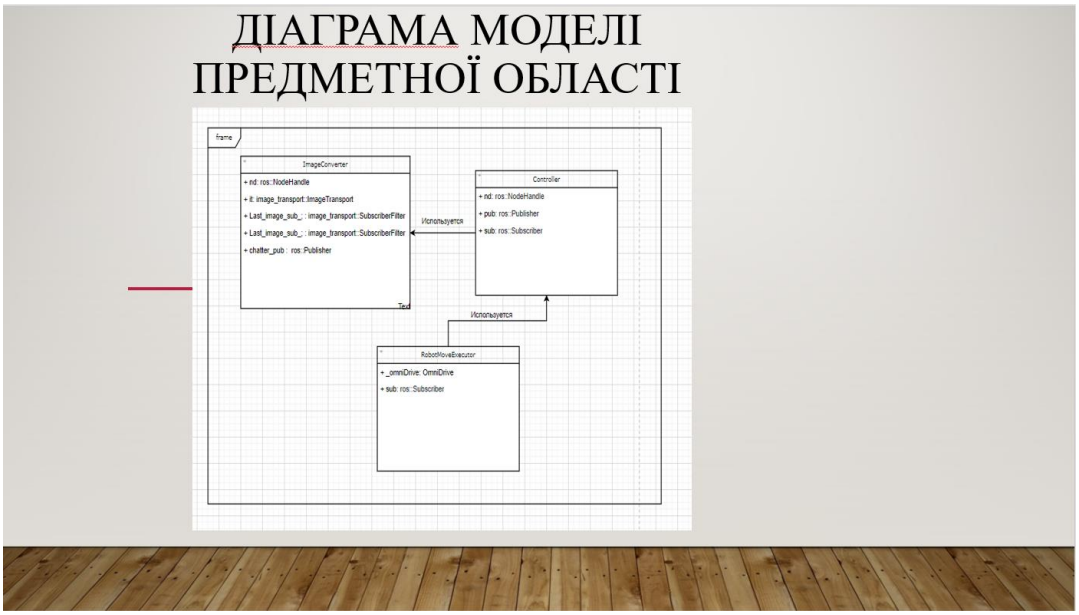
Слайд 4

РОЗРОБЛЕНІ ДІАГРАМИ UML



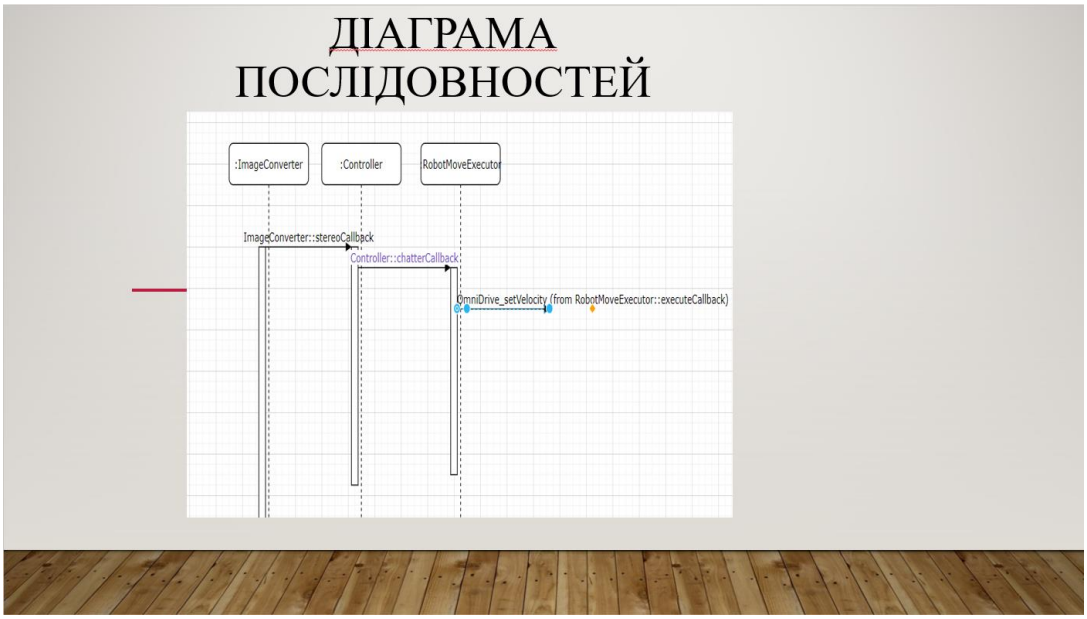
Слайд 5

ДІАГРАМА МОДЕЛІ ПРЕДМЕТНОЇ ОБЛАСТІ



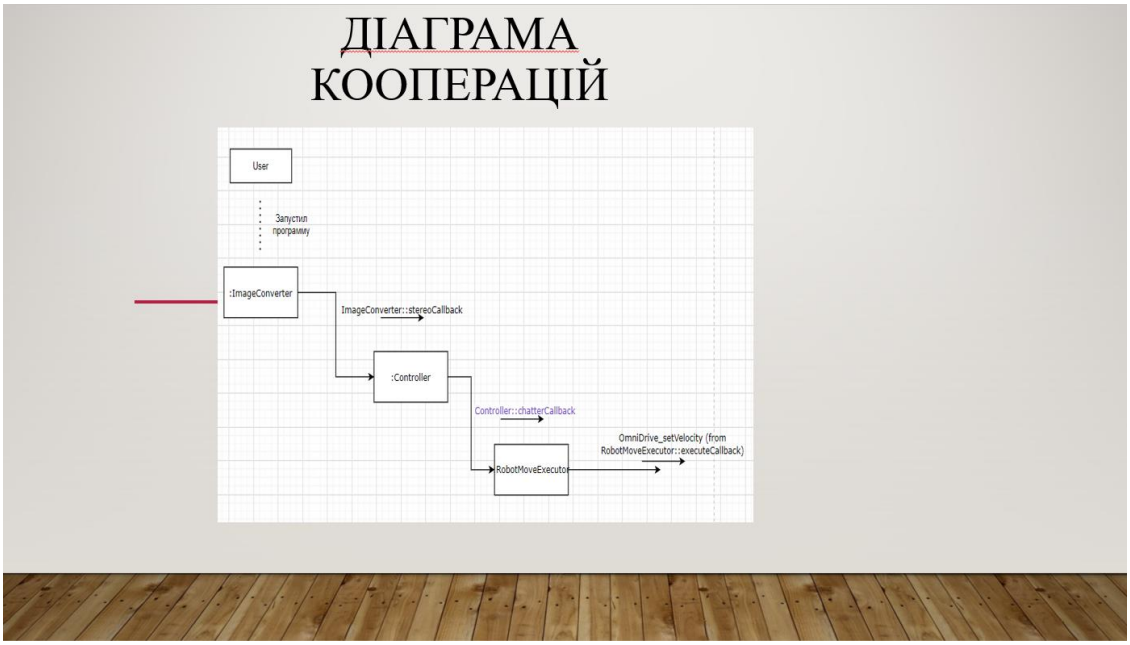
Слайд 6

ДІАГРАМА ПОСЛІДОВНОСТЕЙ



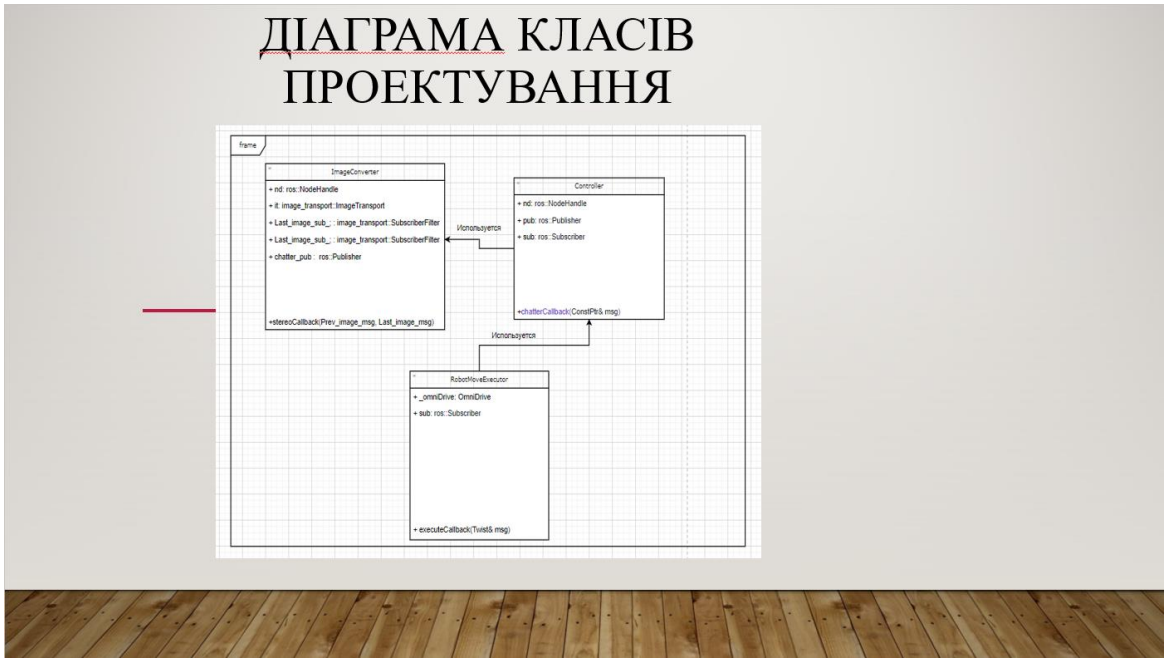
Слайд 7

ДІАГРАМА КООПЕРАЦІЙ



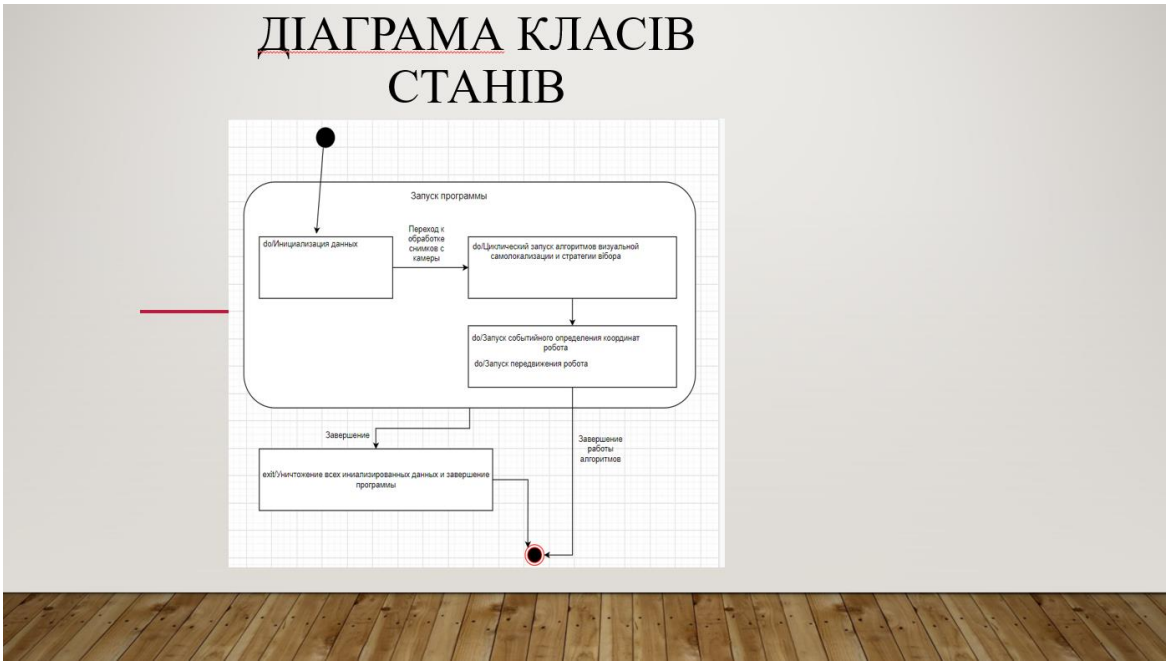
Слайд 8

ДІАГРАМА КЛАСІВ ПРОЕКТУВАННЯ



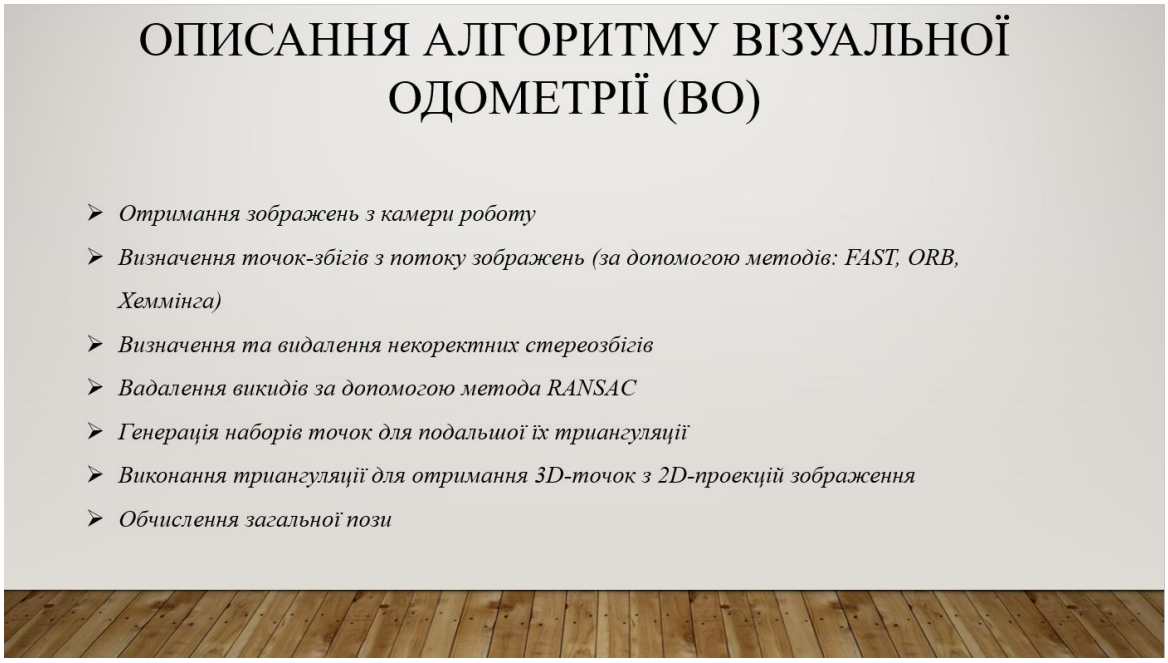
Слайд 9

ДІАГРАМА КЛАСІВ СТАНІВ



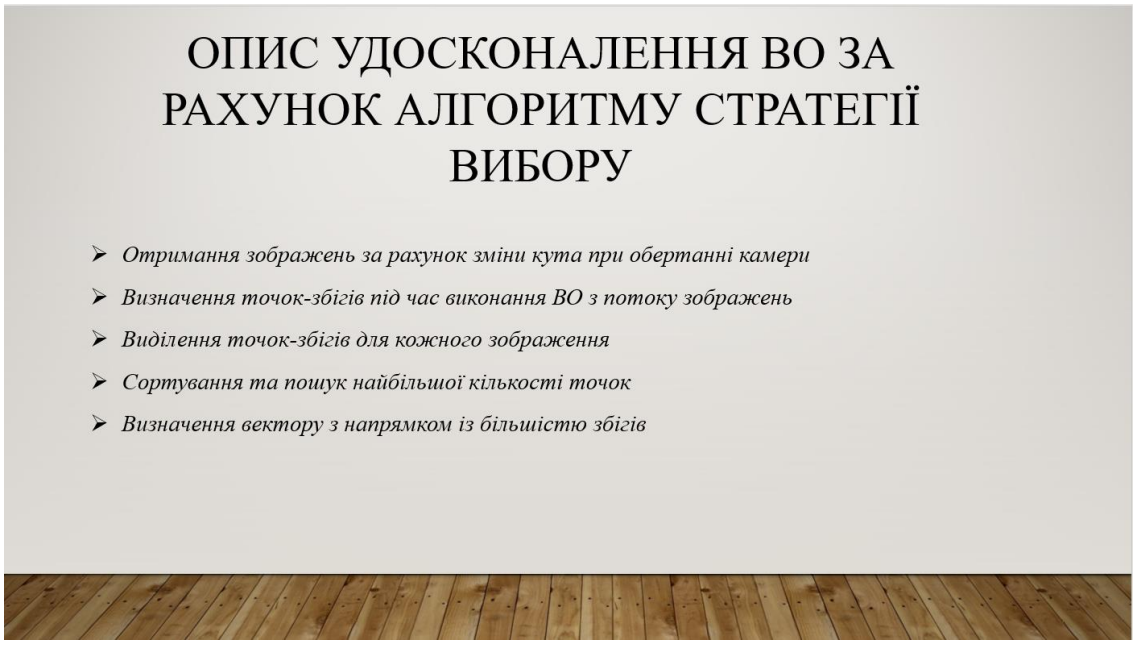
Слайд 10

ОПИСАННЯ АЛГОРИТМУ ВІЗУАЛЬНОЇ ОДОМЕТРІЇ (ВО)



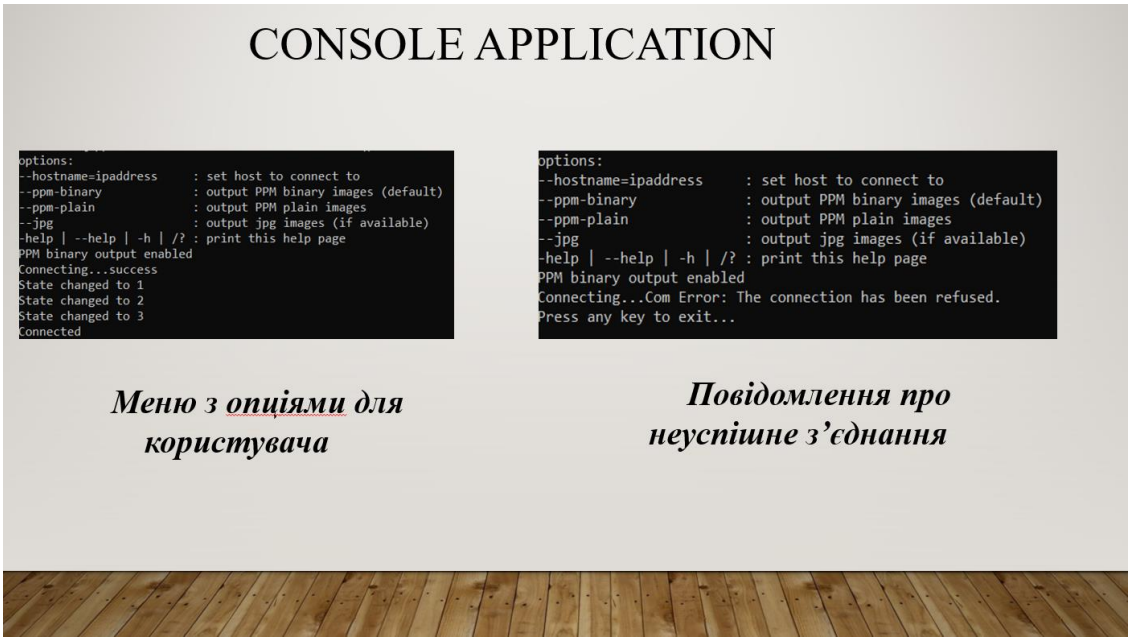
Слайд 11

ОПИС УДОСКОНАЛЕННЯ ВО ЗА РАХУНОК АЛГОРИТМУ СТРАТЕГІЇ ВИБОРУ



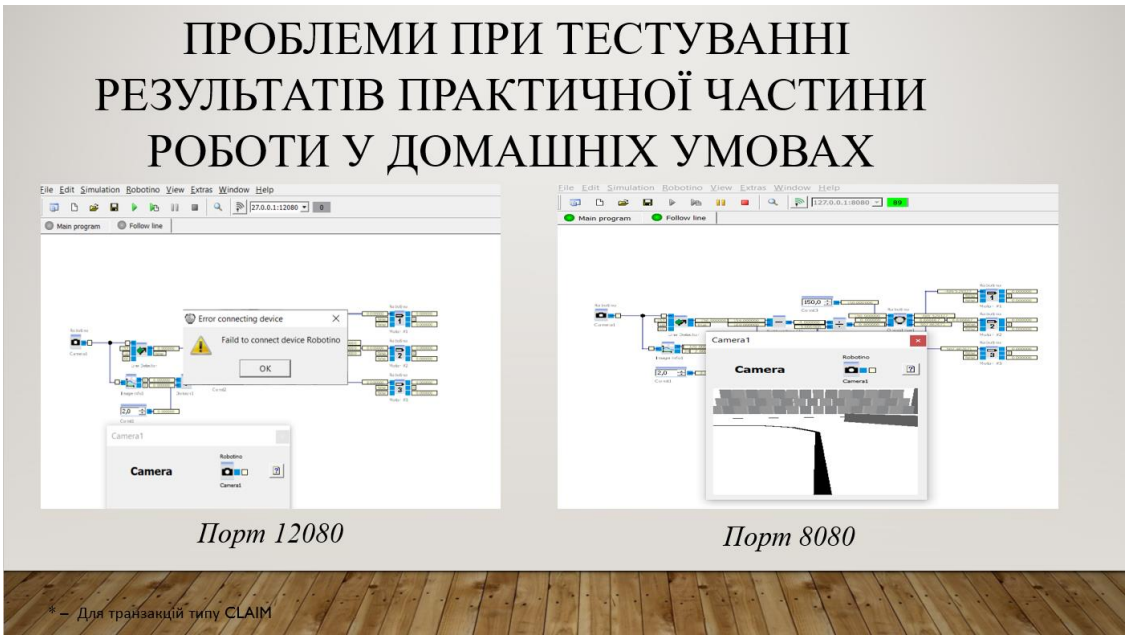
Слайд 12

CONSOLE APPLICATION



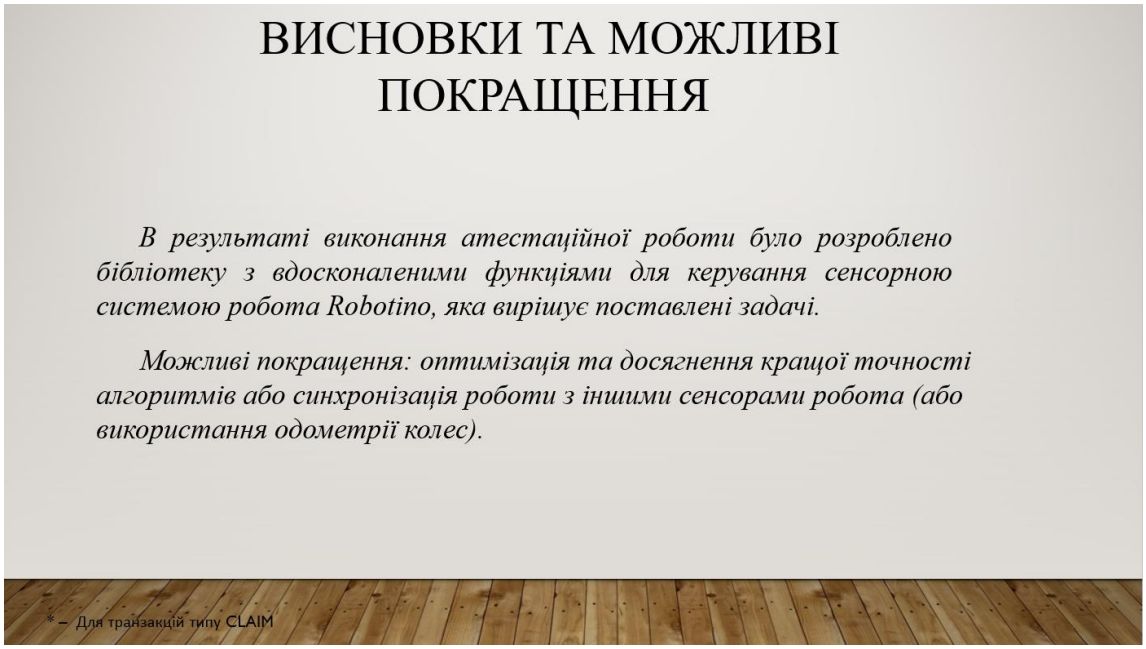
Слайд 13

ПРОБЛЕМИ ПРИ ТЕСТУВАННІ РЕЗУЛЬТАТІВ ПРАКТИЧНОЇ ЧАСТИНИ РОБОТИ У ДОМАШНІХ УМОВАХ



Слайд 14

ВИСНОВКИ ТА МОЖЛИВІ ПОКРАЩЕННЯ



Слайд 15

ДЯКУЮ ЗА УВАГУ

ДОДАТОК В

Відомість атестаційної роботи

№	Позначення	Найменування	Дод. відомості
		Текстові документи	
1	ГЮИК.507160.007 ПЗ	Пояснювальна записка	115 с., ф. А4
		Додаткові матеріали	
2		Демонстраційний матеріали	15 с. у форматі
			*.ppt

Зм.	Арк.	№ докум.	Підпис	Дата	ГЮИК.504300.004 ВД			
Розробив		Рижов В.Б.			Розробка програмної бібліотеки для моделювання функцій сенсорної системи мобільного робота Festo Robotino Відомість атестаційної роботи	Літера	Аркуш	Аркушів
Перевірив		Сінотін А.М.						1
						ХНУРЕ кафедра КІТАМ		
Н.контроль		Яшков І.О.						
Затвердив		Невлюдов І.Ш.						