Development of robot manipulator and motion control using inverse kinematics for robotized vegetable harvesting

Armands Kviesis, Vitalijs Osadcuks

Latvia University of Agriculture, Liela Str.2, Jelgava, LV 3001, Latvia
armands.kviesis@inbox.lv, vitalijs.osadcuks@llu.lv

Abstract: *Robot manipulators can be mentioned as one of technological developments in agricultural sector, which can be used in fruit, vegetable harvesting. It is important to do the harvesting very precisely. Precision does not start at the point where manipulator has to cut the fruit or vegetable, but it starts form the beginning, that is, from the right manipulator's design choice to its correct control. Therefore principles for manipulator development were described and methods for solving inverse kinematics problem were compared and results – analyzed; main task – precise pointing at vegetable – was performed. During experiments it was stated that some methods had singularities. Tests with real vegetable showed that there are factors that affect vegetable detection.*

Keywords: robot manipulator, inverse kinematics, vegetable detection.

Introduction

It is well-known that in agricultural sector technological developments can reduce production costs (Hayashi et al., 2005). Nowadays development of robotic systems in agriculture has experienced an increased interest (Pedersen et al., 2008). In the field of robotized fruit and vegetable harvesting, the whole process can be divided into two main stages: recognition and the actual harvesting. Usually fruit and vegetable recognition is done using image processing, but harvesting - using robot manipulator.
When developing a robot manipulator, various factors must be considered, because they can have significant impact on development costs and also increase the complexity of the control system.
Motion of manipulator, like other rigid body motion when the causes of motion are not considered, is described using kinematics. Depending on what the problem is, forward or inverse kinematics is used. Forward kinematics problem is easier to solve, than inverse kinematics, because forward kinematics is based on the manipulation with the structure that is done by changes of the joint angles inside the controlled structure, but inverse kinematics is based on the direct manipulation with the end of the structure and the joint angles are derived from changes of the end of the structure (Bařinka and Berka, 2002). In case with vegetable harvesting, inverse kinematics method is used, because the target coordinates are known, but the values of robot manipulator joint angles are not. Many scientific articles provide information about several methods for solving inverse kinematics problem. These methods are such as inverse Jacobian, Jacobian transpose, analytic method and other. Different methods have been tested and compared by (Aristidou and Lasenby, 2009; Buss, 2004; Bařinka and Berka, 2002). But not all inverse kinematics methods can be used at every task.
The aim of the research is to develop robot manipulator and its motion control for vegetable harvesting. This paper focuses on robot manipulator development; analysis and comparison of inverse kinematic methods to ensure manipulator's ability to reach the target object. Target object coordinates were obtained by camera, using image processing.

Materials and methods

Manipulator's design and build

Before building manipulator, its design and materials must be chosen according to the task, this manipulator will perform. There are several manipulator design types such as polar coordinate type, articulated type, cylindrical coordinate type, Cartesian coordinate type and SCARA type (Sakai et al., 2008). Different factors characterize manipulator design. The basic mechanism of a manipulator is defined by its degrees of freedom, the type of joint, link length and offset length (Kondo and Ting, 1998).
For the task, to point manipulator's end-effector at target object, design, similar to articulated type, was chosen (Fig. 1).

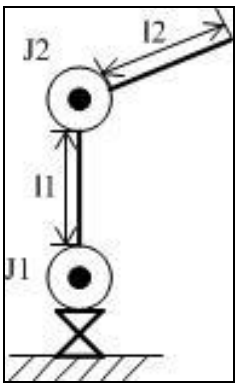

Fig. 1. **Selected manipulator design:** l1, l2 – links, J1, J2 – joints.

As it is shown in the figure, manipulator has three DOF or degrees of freedom. A degree of freedom is a joint on the arm, a place where it can bend or rotate or translate (Robot arm tutorial). All three DOFs in this case are rotational.

Different types of motors such as servo, DC, stepper, can also be used as manipulator's joints. In this research servo motors were used due to their built-in capability of moving to precise position. It is very important to choose the right motors, because if the motors are not suitable for the specific task, the whole manipulator can be damaged in the worst case scenario.

One of the main things that must be taken into consideration, when choosing a motor, is its torque. To find out the torque that is required to lift manipulator's link or "shoulder", it must be calculated (Robot statics). Torque can be calculated using formula (1):

$$M = F * l \tag{1}$$

where M – motor torque, Nm;
F – force, N;
l – distance, m.

To calculate torques, the manipulator must be stretched to its maximum length horizontally (Fig. 2), because then manipulator's motors torque will be the highest.

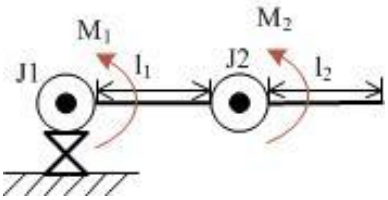

Fig. 2. **Manipulator position for motor torque calculation:** M_1, M_2 – torques, l_1, l_2 – links, J1, J2 – joints.

Torque for motor 1 is calculated in (2):

$$M_1 = \frac{l_1}{2} * G_1 + l_1 * G_m + \left(l_1 + \frac{l_2}{2}\right) * G_2 + (l_1 + l_2) * G_o \tag{2}$$

where M_1 – torque of motor 1, Nm;
l_1 – length of link 1, m;
l_2 – length of link 2, m;
G_1 – weight of link 1, N;
G_2 – weight of link 2, N;
G_m – weight of motor 2, N;
G_o – weight of object, N.

Formula above (2) can be used, if manipulator's link is homogeneous rod. If it is not, then center of mass should be determined differently (depending on the rod). And then the calculation should look like (3):

$$M_1 = a * G + l_1 * G_m + \left(l_1 + \frac{l_2}{2}\right) * G_2 + (l_1 + l_2) * G_o \tag{3}$$

where M_1 – torque of motor 1, Nm;
a – center of mass, m;
G – weight, N;
l_1 – length of link 1, m;
l_2 – length of link 2, m;
G_m – weight of motor 2, N;
G_2 – weight of link 2, N;
G_o – weight of object, N.

Torque for motor 2 is calculated in (4):

$$M_2 = \frac{l_2}{2} * G_2 + l_2 * G_o \quad (4)$$

where M_2 – torque of motor 2, Nm;
l_2 – length of link 2, m;
G_2 – weight of link 2, N;
G_o – weight of object, N.

The same thing about homogeneous rod applies to torque for motor 2. As we can see, there is weight G_o which is some object's weight. It was added for safety reasons, pretending that there is some object at the end of link 2. In this case, even though there is no object intended to be, this gives confidence about that the robot manipulator will not collapse by just lifting only these two links and one motor.

By these formulas (2-4) we can conclude that, if more DOFs are added, more complicated these formulas become, more torque is required for motors. Motor torque is also affected by link properties – for lighter weight, and shorter links smaller torque is required.

After torque calculation three motors were chosen and the manipulator was developed. Those motors are:

- two HD-6001HB,
- HITEC HS-311.

Motor physical control was ensured using microcontroller Atmega8.

Methods for solving inverse kinematics problem

Many authors divide inverse kinematics methods in some kind of groups. Methods are divided in algebraic and iterative groups by (Bařinka and Berka, 2002). Algebraic method group consist of a method which is based on basic trigonometry. The second group, iterative, consists of methods such as inverse Jacobian, cyclic coordinate descent (CCD), genetic programming, Jacobian transpose and other. These methods are called iterative, because many steps are required to solve the problem. Based on conclusions by (Bařinka and Berka, 2002), methods, such as algebraic or analytic, inverse Jacobian, Jacobian transpose, Jacobian pseudoinverse, were implemented, compared and analyzed.

Analytic method

As mentioned before, algebraic or analytic method is based on basic triangle calculations. Fig. 3 shows a situation where manipulator has reached target point.

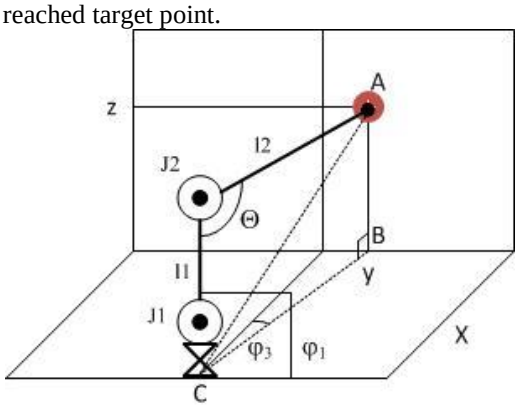

Fig. 3. **Analytic method's interpretation in 3 dimensions:** X, y, z – target object coordinates, φ_1, φ_3 – DOF angles, l1, l2 – links, J1, J2 – joints, Θ - angle between l1 and l2.

For analytic solving, first we need to create a triangle. In Fig.3 the triangle is *ABC*. When it's done, it can be transformed into two dimensional plane (Fig. 4).

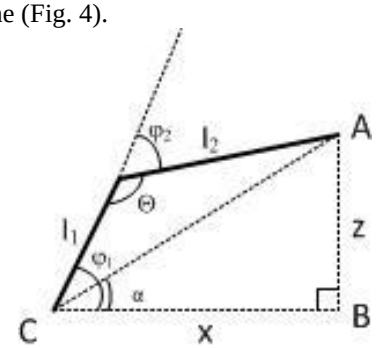

Fig. 4. **Analytic method's interpretation in 2 dimensions:** x – distance to object by *x* axis, z – distance to object by *z* axis, α – angle in triangle ABC, l_1, l_2 – links, φ_1, φ_2 – l_1 and l_2 angles respectively, Θ - angle between l_1 and l_2.

As we can see in Fig. 4, there are two triangles, where two sides are known, but the third is not. *AC* is calculated using Pythagorean Theorem. Angles such as Θ and φ_1 are calculated using the law of cosine. The equations for φ_1 and Θ are shown below:

$$\varphi_1 = \arccos\left(\frac{x}{\sqrt{x^2+z^2}}\right) + \arccos\left(\frac{x^2+z^2+{l_1}^2-{l_2}^2}{2\sqrt{x^2+z^2}*l_1}\right) \quad (5)$$

where φ_1 – link 1 angle, degrees ^{0};
x – distance to object by x axis, m;
z – distance to object by z axis, m;
l_1 – length of link 1, m;
l_2 – length of link 2, m.

$$\theta = \arccos\left(\frac{{l_1}^2+{l_2}^2-x^2-z^2}{2*l_1*l_2}\right) \quad (6)$$

where Θ – angle between l_1 and l_2, degrees ^{0};
x – distance to object by x axis, m;
z – distance to object by z axis, m;
l_1 – length of link 1, m;
l_2 – length of link 2, m.

Angle Θ is angle between links l_1 and l_2, but that is not what is needed. To calculate value of φ_2, Θ must be subtracted from 180^0 (adjacent angle feature).
The equation for solving φ_2 is shown in (7).

$$\varphi_2 = 180^0 - \theta \quad (7)$$

where φ_2 – link 2 angle, degrees ^{0};
Θ – angle between l_1 and l_2, degrees ^{0}.

To solve the third angle that represents by how much to turn the whole manipulator, equation (8) was used.

$$\varphi_3 = \operatorname{arctg}\left(\frac{(C-y)}{X}\right) \quad (8)$$

where φ_3 – manipulators turn angle, degrees ^{0};
y – object coordinate by y axis, m;
X – object coordinate by x axis, m;
C – center position by y axis, m.

It should be noted, that analytic method can be used when number of DOFs is small, for example two. In this case there were three DOFs, but this method still worked. It was because the third DOF, the base, simply turns the whole manipulator, so it was easy to solve. But when the number of DOFs is higher, these angles cannot be solved in a trivial way. Therefore more sophisticated approaches are necessary (Bařinka and Berka, 2002). Analytic method also is described by author (Yetim, 2009).
Most popular methods for inverse kinematics problem solving are those where Jacobian matrix is taking part. The Jacobian matrix $J(e,\varphi)$ shows how each component of e varies with respect to each joint angle (Rotenberg, 2005). A Jacobian matrix is nothing more than a matrix of partial derivatives of the entire chain system relative to the end-effectors (Aristidou and Lasenby, 2009). For a two dimensional robot manipulator the basic Jacobian matrix can be written as follows (9):

$$J(e,\Phi) = \begin{bmatrix} \frac{\partial e_x}{\partial \varphi_1} & \frac{\partial e_x}{\partial \varphi_2} \\ \frac{\partial e_y}{\partial \varphi_1} & \frac{\partial e_y}{\partial \varphi_2} \end{bmatrix} \quad (9)$$

where J – Jacobian matrix;
e – end-effectors' position;
Φ – joint angles.

End-effector's position is determined using equations that correspond to each end-effector's coordinate in Cartesian coordinate system. Fig. 5 shows a schematic representation how the situation (manipulator and target object emplacement) is looking like. Difference from analytic method – this method starts solving form beginning, taking into account the starting end-effector position, while in analytic method the whole process was started, assuming that manipulator had reached the object.

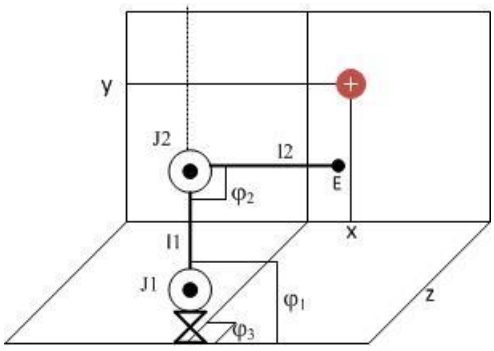

Fig. 5. **Manipulator and target object emplacement example:** φ_1, φ_2, φ_3 – DOF angles, E – end-effector, x,y,z – object coordinates, l1, l2 – links, J1, J2 – joints.

In Fig.5 we can see the starting position of manipulator. The end-effector's coordinates are calculated by equations below:

$$e_x = (l_1 cos\varphi_1 + l_2 \cos(\varphi_1 + \varphi_2))cos\varphi_3 \quad (10)$$

where e_x – end-effector's coordinate by *x* axis, m;
l_1 – length of link 1, m;
l_2 – length of link 2, m;
φ_1 – link 1 angle, degrees ^{0};
φ_2 – link 2 angle, degrees ^{0};
φ_3 – manipulators turn angle, degrees ^{0}.

$$e_y = (l_1 cos\varphi_1 + l_2 \cos(\varphi_1 + \varphi_2))sin\varphi_3 \quad (11)$$

where e_y – end-effector's coordinate by *y* axis, m;
l_1 – length of link 1, m;
l_2 – length of link 2, m;
φ_1 – link 1 angle, degrees ^{0};
φ_2 – link 2 angle, degrees ^{0};
φ_3 – manipulators turn angle, degrees ^{0}.

$$e_z = l_1 sin\varphi_1 + l_2 sin\,(\varphi_1 + \varphi_2) \quad (12)$$

where e_z – end-effector's coordinate by *z* axis, m;
l_1 – length of link 1, m;
l_2 – length of link 2, m;
φ_1 – link 1 angle, degrees ^{0};
φ_2 – link 2 angle, degrees ^{0};

The next step is to put these equations in Jacobian matrix and calculate the derivatives resulting in a matrix that can be used in selected methods.

Inverse Jacobian method

The problem, that must be solved, has a non-linear nature. Thus it cannot be solved directly, it needs to be linearized. Linearization process can be done by (13):

$$\Delta e \approx J\Delta\theta \quad (13)$$

where Δe – change in end-effector position, m;
J – Jacobian matrix;
$\Delta\Theta$ – change in joint angles, degrees ^{0}.

It tries, through *small* changes in the vector of angles $\vec{\theta}$ of the manipulator, to introduce small changes in the position of the end-effector to finally reach the desired position (Park). The change in joint angles using inverse Jacobian method, can be written like (14):

$$\Delta\theta = J^{-1}\Delta e \quad (14)$$

where $\Delta\Theta$ – change in joint angles, degrees ^{0};
J^{-1} – inverse of Jacobian matrix;
Δe – change in end-effector position, m.

As described by (Aristidou and Lasenby, 2009), there are cases where Jacobian matrix cannot be inverted. This must be taken into account when trying to implement inverse Jacobian method.

Jacobian transpose method

This method is very well explained by (Aristidou and Lasenby, 2009). Basically, this method takes away problem that exists in inverse Jacobian method – possibility that matrix could not be inverted. It is so, because in

Jacobian transpose method there are no such thing as matrix inversion. Matrix transposition is performed instead of inversion. Thus this method is considered to be much faster than the inverse Jacobian method. This method is defined in (15):

$$\Delta\theta = \alpha J^T \Delta e \tag{15}$$

where $\Delta\Theta$ – change in joint angles, degrees ^{0};
α – value, usually $0 \leq \alpha \leq 1$;
J^T – transposition of Jacobian matrix;
Δe – change in end-effector position, m.

Jacobian pseudoinverse method

In pseudoinverse method change in joint angles can be expressed like (16):

$$\Delta\theta = J^+ \Delta e \tag{16}$$

where $\Delta\Theta$ – change in joint angles, degrees ^{0};
J^+ – pseudoinverse of Jacobian matrix;
Δe – change in end-effector position, m.

The solution for J^+ is described with formulas by (Yao, 2009). The main formula, which explains how to compute J^+ is shown in (17):

$$J^+ = (J^T J)^{-1} J^T \tag{17}$$

where J^+ – pseudoinverse of Jacobian matrix;
J^T – transposition of Jacobian matrix;
J – Jacobian matrix.

Authors (Aristidou and Lasenby, 2009) claim that this method has its downsides: "The pseudo-inverse method is widely discussed in the literature, however it often performs poorly because of its instability near singularities".
For Inverse Jacobian, Jacobian transpose and Jacobian pseudoinverse method the algorithm is very similar. The only difference is the way, how $\Delta\Theta$ is being calculated. Activity diagram is shown in Fig. 6.

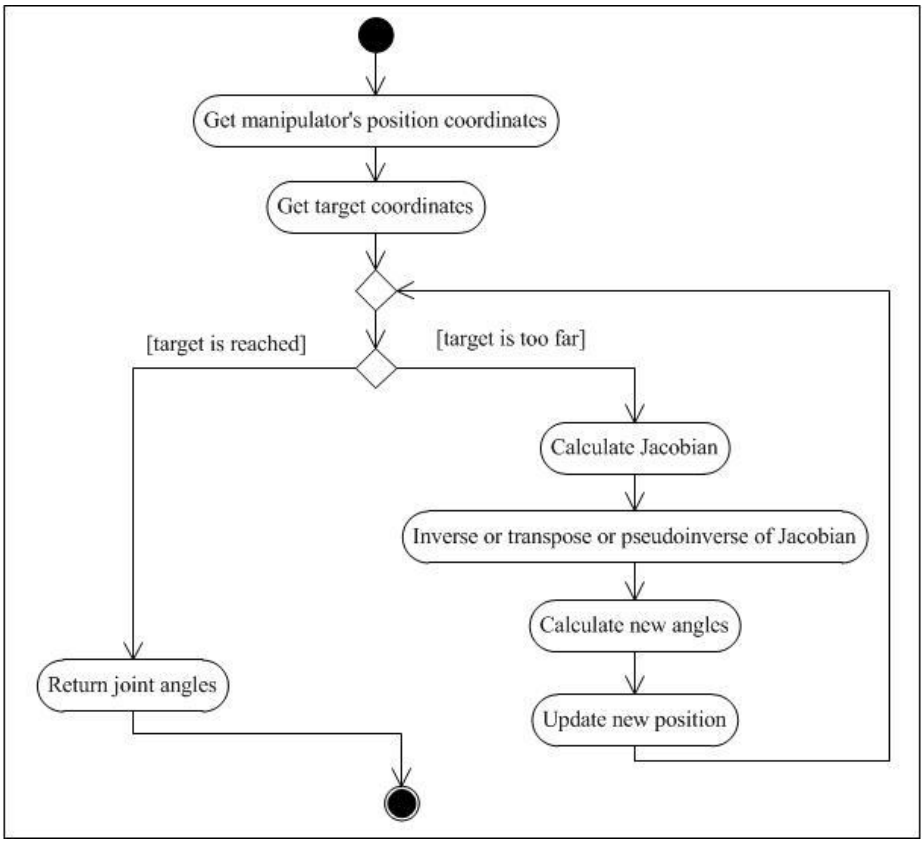

Fig. 6. **Iterative methods activity diagram.**

Usage of chosen methods

These four methods, mentioned above, were implemented into a program for vegetable recognition. The program was developed using C# programming language. For image processing and matrix calculations AForge.NET Framework was used. Images were captured using web camera with these parameters:

- sensor resolution: 1.3Mpixel,

- image sensor technology: CMOS,
- maximum video resolution: 640x480 (used in experiments: 320x240),
- maximum frame capture rate: 30fps (640x480).

Implemented methods were compared and analyzed. Results of that analysis are described in section "Results and discussion".

Results and discussion

During this research a program was developed using C# programming language. This program ensures:

- image processing,
- manipulator's angles calculations,
- sending information to microcontroller for manipulator's control.

Because of images and recognition being processed continuously and robot manipulator is controlled as soon as the angles are calculated, program was considered to be a real-time.

Comparison criteria were set for the chosen methods. It was considered, that one of the preconditions of these methods to use them in a real-time program is their execution time. If it is taking too long to calculate the joint angles, then the program will work poorly, because it must wait, while the calculation stops, to be able to turn the manipulator.

Since the manipulator is aimed to vegetable harvesting, precision is very important factor. Thus the next criterion was how precise the methods can calculate joint angles, in order to point the manipulator's end-effector to the target.

So the criteria were:

- execution time, ms;
- position error, %;

Experiments were made with C# code on 2.6GHz Athlon 64 X2.

For time calculation a time was fixed when a method starts angle calculation, and the end time was fixed when calculation was finished, so the final execution time is difference in milliseconds between these two times. This was done several times and the results were obtained with average values. The determined average execution times are shown in Table 1.

Table 1

Inverse kinematics methods average execution time

Method name	Time, ms	Iterations
Inverse Jacobian	0.59	316
Jacobian transpose	0.07	35
Jacobina pseudoinverse	7.1	507
Analytic method	0.0007	1

From Table 1 it can be concluded, that the fastest execution time had analytic method. It is so, because, this method does not belong to iterative methods group. Thus it was expected to be so. Jacobian pseudoinverse method's execution time was worst and was quite different from other.

To determine position error, a marker was attached to manipulator's end-effector. For targets red squares drawn on a paper was used (Fig. 7).

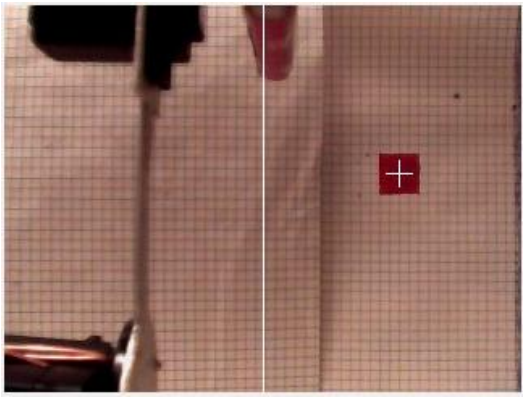

Fig. 7. **Position error detection environment.**

Error was determined by measuring the distance between manipulator's marked point and the actual target's center point, at the end total error by *x* and *y* coordinate was summed. A total of three attempts were made for each method, where in each attempt target was at different position. At the end, the average error of each method was calculated and expressed in percentages. Results are shown in Fig. 8.

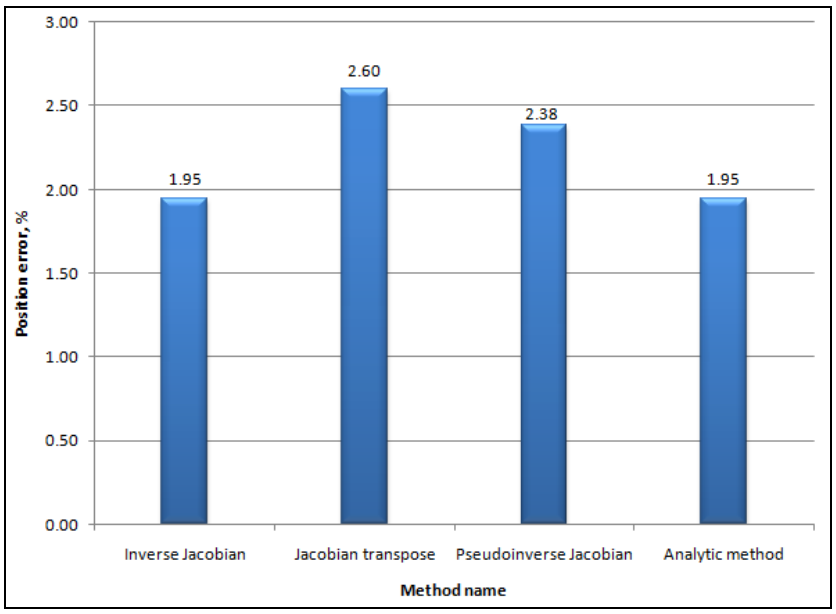

Fig. 8. **Average position error from three attempts.**

As it can be seen, the maximum precision error does not exceed 2.6%. Best results were achieved, when using inverse Jacobian and analytic method. Author (Buss, 2004) claimed, that the Jacobian transpose had the advantage of being fast, but of poor quality, the same happened here – Table 1 shows that Jacobian transpose is the fastest between iteration methods, but it has the biggest position error (Fig. 8).

During the experiments, only once a singularity, mentioned by (Aristidou and Lasenby, 2009), was detected by inverse Jacobian method, where it did not find a solution for one target position.

Referring to the paper by authors (Bařinka and Berka, 2002) about Jacobian transpose method, that some troubles like in the Jacobian inversion are still there, e. g. singularities, it must be said, that there was observed a singularity, when choosing α value – this method did not perform its task for some α values.

It is important to mention, that position error can be affected by how precise target object can be recognized and how well scaling process is done. While the experiments were performed, the error by image processing fluctuated due to light change. But these fluctuations were not very large, mostly under 0.7%. Error levels expressed in percentages at each attempt made by image processing are shown in Fig. 9.

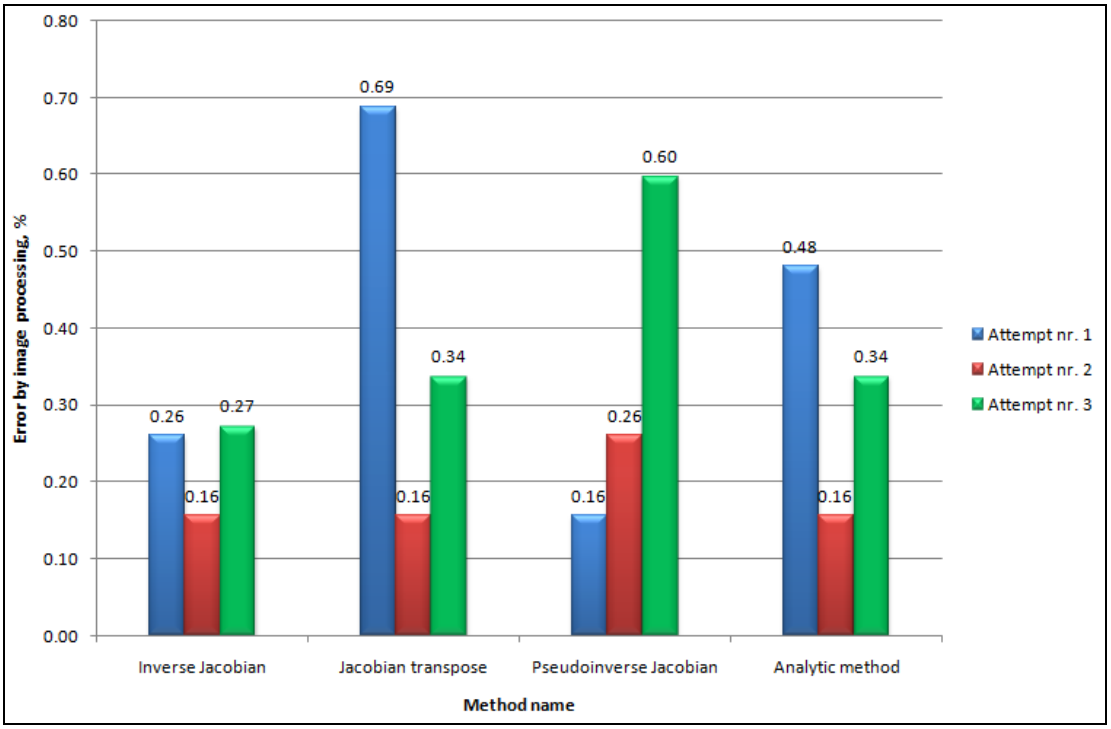

Fig. 9. **Image processing error in each attempt.**

Position error is also affected by how precise the manipulator is built.
After the analysis of inverse kinematics methods, analytic method was chosen as the suitable one and tested with real vegetable – tomato. This method was chosen because of its fast execution time and quite good precision, despite of the error made by image processing, which was higher than, when inverse Jacobian method was tested (both has the same error level though).
When the recognition and manipulator's positioning process was applied for tomato detection, some factors that affected the whole process were observed. Experiments with tomato revealed that the shape of tomato needs to take into account, because when testing the precision of inverse kinematics methods, two dimensional targets were used, but tomato is not in two dimensions. Thus during experiments often was observed situation, when manipulator hit the tomato instead of pointing to it at very close distance.
Next thing that was observed was light reflection of tomato, due to its smooth surface. In some cases this gave an impact to image processing, resulting in inaccurate coordinate detection thus affected further recognition process.
In Fig.10 tomato detection is shown. As it can be seen from side view, manipulator's end-effector does not point to tomato very precisely, reasons for that have been mentioned above.

Fig. 10. **Tomato detection:** 1. – tomato recognition (web camera view), 2. – manipulator pointing to tomato (web camera view), 3. – whole situation from side view.

Conclusion

This paper described manipulator's development principles and analysis of several methods that can be used to solve inverse kinematics problem. After inverse kinematics methods comparison and analysis, it was concluded, that the best results were obtained with analytic method, because of its very fast execution time and best error level. Analytic method's execution time was 0.0007 ms, and precision error was 1.95%. Pseudoinverse Jacobian method had the worst execution time – 7.1 ms, but Jacobian transpose method had the worst precision error – 2.6%. Analytic method was chosen to use with real vegetable recognition.
Tests with tomato showed that there are some factors that made an impact to recognition process. Thus using only image processing and methods for manipulator positioning is not enough for complete vegetable detection. To use this recognition system in a field, manipulator should be equipped with extra sensor/-s.

References

Aristidou, A. and Lasenby, J., 2009. Inverse Kinematics: a review of existing techniques and introduction of a new fast iterative solver. Cambridge University Engeneering Department. CUED/F-INFENG/TR-632. 74 p.

Bařinka, L. and Berka, R., 2002. Inverse Kinematics – Basic Methods. Dept. of Computer Science & Engineering. pp. 10

Buss, S.R., 2004. Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods. Department of Mathematics; University of California, San Diego. 19 p.

Hayashi, S., Ota, T., Kubota, K., Ganno, K., Kondo, N., 2005. Robotic Harvesting Technology for Fruit Vegetables in Protected Horticultural Production. *FRUTIC 05*, Montpellier France. pp. 227 – 236

Kondo, N. and Ting, K.C., 1998. Robotics for Plant Production. *Artificial Intelligence Review* 12(1-3). pp. 227-243

Park, D. Inverse Kinematics. Computer Graphics, Department of Computer Science, University of Buenos Aires. 9 p.

Pedersen, S.M., Fountas, S., Blackmore, S., 2008. Agricultural Robots – Applications and Economic Perspectives. Service Robot Applications. I-Tech Education and Publishing KG, pp. 369-382

Robot arm tutorial. Available at: http://www.societyofrobots.com/robot_arm_tutorial.shtml, 04.12.2012.

Robot statics. Available at: http://www.societyofrobots.com/mechanics_statics.shtml#momentarms, 04.12.2012.

Rotenberg, S., 2005. Inverse Kinematics (part 2). Available at: http://graphics.ucsd.edu/courses/cse169_w04/CSE169_12.ppt, 04.12.2012

Sakai, S., Iida, M., Osuka, K., Umeda, M., 2008. Design and control of a heavy material handling manipulator for agricultural robots. *Autonomous Robots* 25(3). pp. 189 – 204

Yao, M., 2009. Mathematics for Inverse Kinematics. 15-464: Technical Animation. 30 p.

Yetim, C., 2009. Kinematic Analysis for Robot Arm. Yildiz Technical University, Electrical and Electronics faculty, Department of Computer Engineering. Istanbul. 34 p.