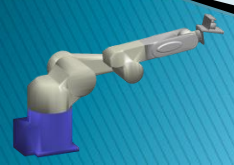
Week 9

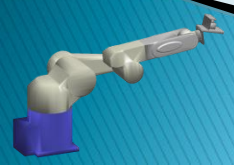
Schedule

- Quiz 4 (with your Tutor):
  - Individual (35 mins)
  - Team (25 mins)
  - Review (5 mins)
- Gavin
  - Lab Assignment 2 update (5 mins)
  - Future plans (5 mins)
- Sheila
  - Revise & discuss Lab 8 (5 mins)
  - Lab 9 Exercise Introduction (5 mins)
  - Work on Lab 9 Exercises (≈1.5 hours)



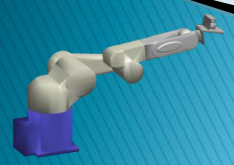
Overview of Week 9: Quiz 4

- In total worth 5%
- Questions have different weighting
- Individual quiz
  - worth 4%
  - will go from lab start time for 35 minutes
  - 11 questions (approx. 1 question from each category)
  - +1 share your screen question
  - No talking
- Group quiz
  - worth 1%
  - will start immediately after the individual quiz
  - will go for 25 minutes
  - Groups of 3 people or less
  - 21 questions (approx. 2 question from each category)
  - Lots of talking within group

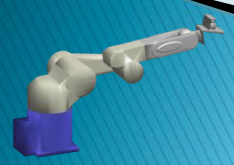


Question Categories

- Manipulability Measure
- Collision Detection Line (basic)
- Collision Detection Points
- Near Singularity 6Dof
- Near Singularity 3Dof Planar
- Resolved Motion Rate Control
- Optimisation
- Collision Detection Line (harder) (**worth more)**
- Visual Servoing (IBVS) (**worth more)**

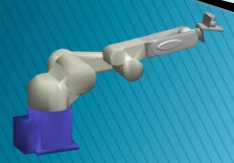


Run Quiz



Code/Model Plagiarism (1 of 2)

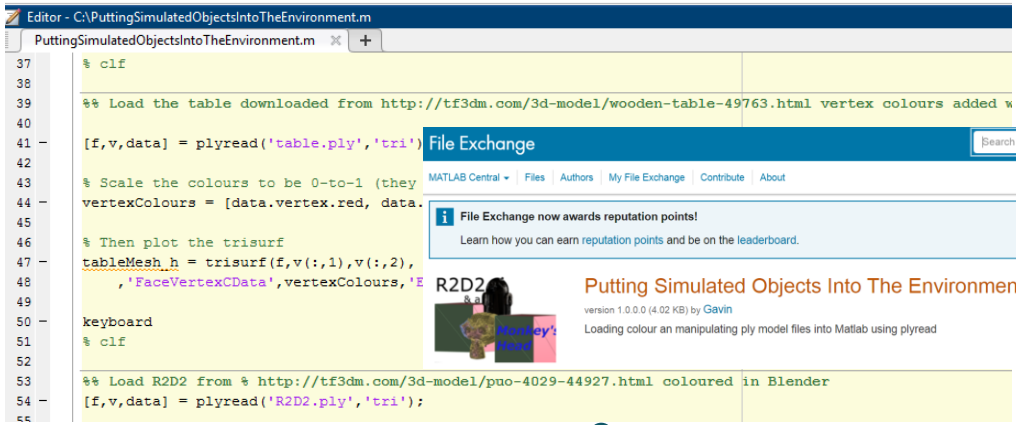
- What is NOT ok?
  - Copying a previous student's models and changing the filename (to avoid detection)
  - Copying a previous student's code and
    - doing find/replace so the variable/ function names are different
    - Just adding more comments to the original code
    - Minor reordering of code blocks to try to avoid detection
- Detection
  - Turnitin on code submissions is easy to get around
  - However, I have several Matlab scripts, Excel macros and other tools to compare files and to find big patches of code similarity
  - Also, I have run the subject from inception, and the tutors Jonathan and Sheila have been involved from the start too
  - The combination of identical files and code which overlaps (without any fundamental IP addition) plus a difficulty explaining pieces of code, makes plagiarism detection and misconduct proceedings straightforward (but emotionally draining for everyone)
- We care about the subject integrity and learning outcomes!!



Code/Model Plagiarism (2 of 2)

- What is ok?
  - Write your own code from scratch
  - Tell us clearly in the demo that you got someone else's model/code and don't want to be marked on it, but you included it for some reason (e.g. safety)
  - Use our subject's, toolbox's (or an internet source's) code/models with clear reference to it

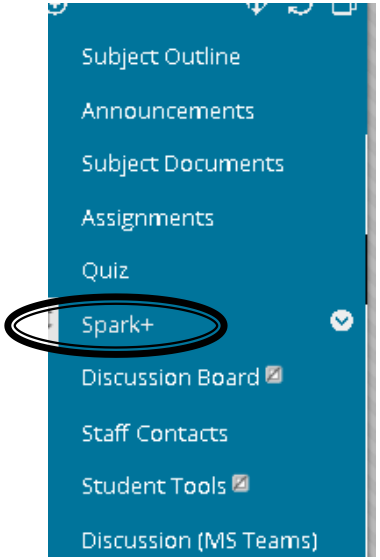
```
%% IsCollision
% This is based upon the output of questions 2.5 and 2.6
% Given a robot model (robot), and trajectory (i.e. joint st
% and triangle obstacles in the environment (faces,vertex,fa
function result = IsCollision(self,robot,qMatrix,faces,verte
```



Let's do better in future!

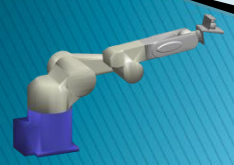
Lab Assignment 2

- Feedback Stage is due after lab class
- Worth 5% of the Lab Assignment
- Upload SWMS and Risk Assessment for real robot to private Teams channel. (1st attempt is marked, but if you want to use a real robot, resubmits are required until adequate)
- Code Repository (i.e. BitBucket/github) commits (Tutors need access)
- Professionally-completed Spark+ short report on participation of other team member(s)
- Marks are distributed as follows:
  - SWMS (1 of 5%) + RA (1 of 5%)
  - Frequency and total number of quality code commits to your code repository (2 of 5%)
  - Professionally-completed Spark+ assessment (1of 5%)
    - Self
    - Peer
    - Comments



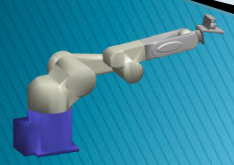
Lab Assignment 2 (hardware)

- Most people will be unable to use the real robots in the lab (marks will be scaled), but there is a possibility
- Once teams complete Safety (RA & SWMS) and upload to teams
- You can then discuss use of robot hardware with the lab staff
- RaspberryPi can be borrowed from lab and taken home or preferably left in locker
- For safety/security, please don't
  - use with wifi
  - connect to the UTS network
  - leave robot unattended
- If Upboard or Robot breaks please tell the lab staff



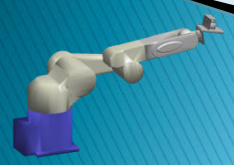
Future Plans

- Week 9:
  - Lab 9 Question 1: Resolved Motion Rate Control in 6DOF
- Week 10 (next week):
  - Continue Lab 9 Questions 2 and 3: Static and Dynamic Torque
  - Assessment Task #4 – Final Report specifications released
- Week 11:
  - Marc & Gavin jointly run the class
  - Joystick input, Human-in-the-loop control: Jogging / Admittance
  - Discussion topics: How will robots affect the future?
  - Assignment 2 help (short video due Friday)
- Week 12:
  - Assignment 2 group demonstration (in class)
  - Final video due Friday
- Week 12-14
  - Assignment 2 individual viva (booking essential)
- Week 14: (assessment week 1)
  - Assessment Task #4 – Final Report Due on Friday

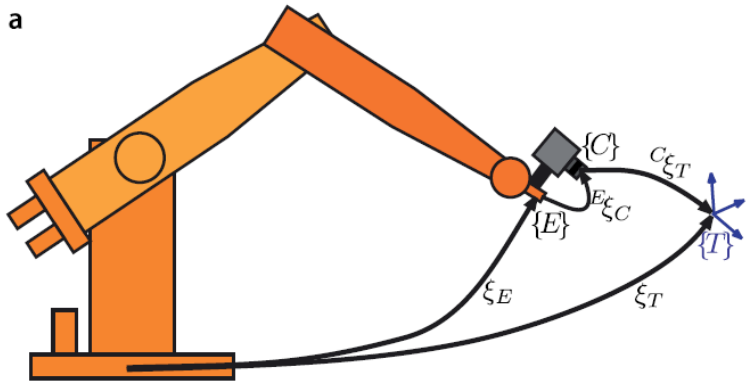


In Resolved Motion Rate Control, which of the following is true

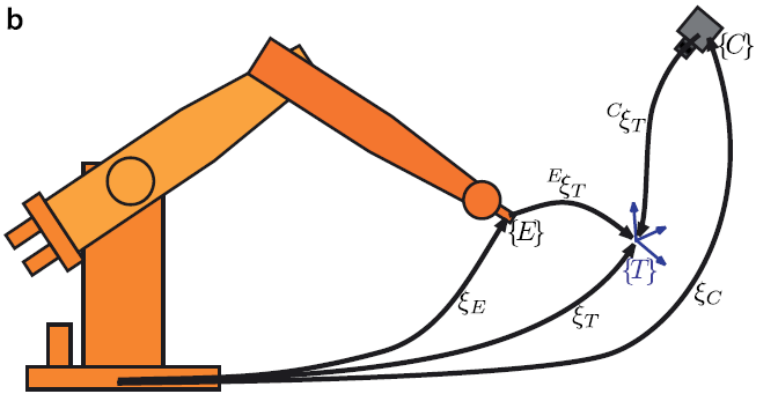
- A robotic system is redundant if the Jacobian has more columns than rows
- Near singular configurations, solutions for joint velocities grow to infinity
- The Jacobian indicates how the end-effector will move as a function of the current joint state



Visual servoing configurations



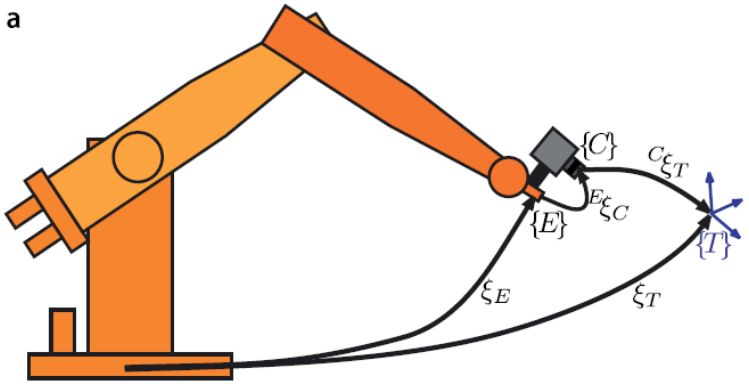
End-point closed-loop
(Eye-in-hand)



End-point open-loop
(Eye-to-hand)

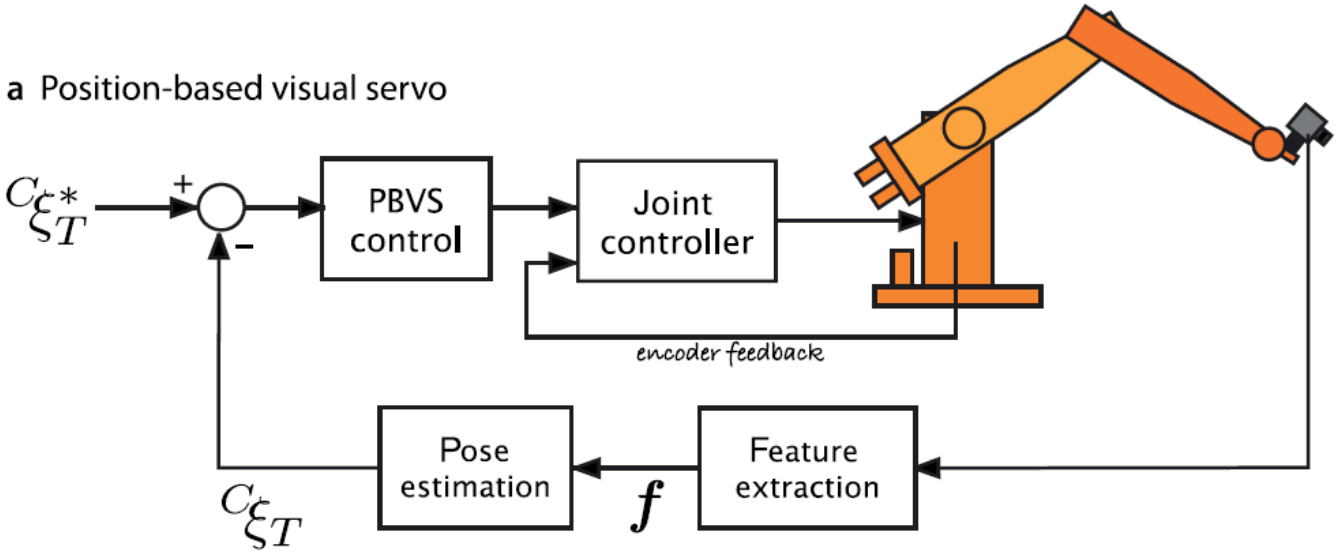
Notation

- Coordinate frames:
  - World (Base)
  - End-effector $\{E\}$
  - Camera $\{C\}$
  - Target $\{T\}$

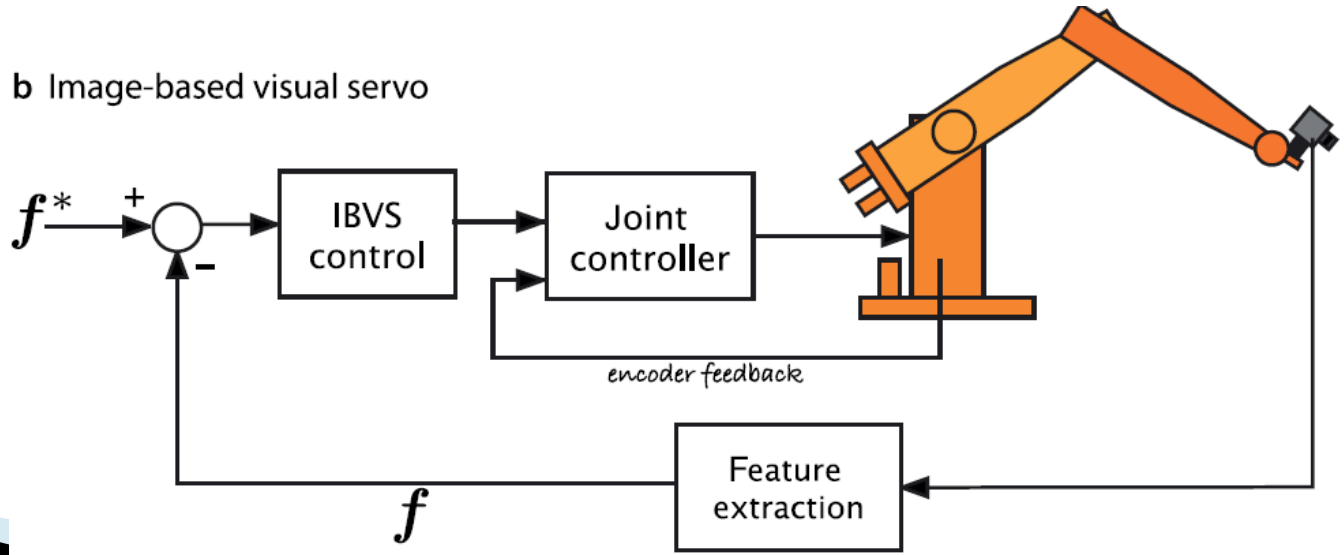


Main Approaches

- PBVS

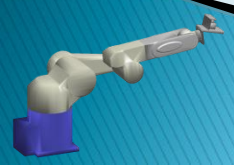


- IBVS

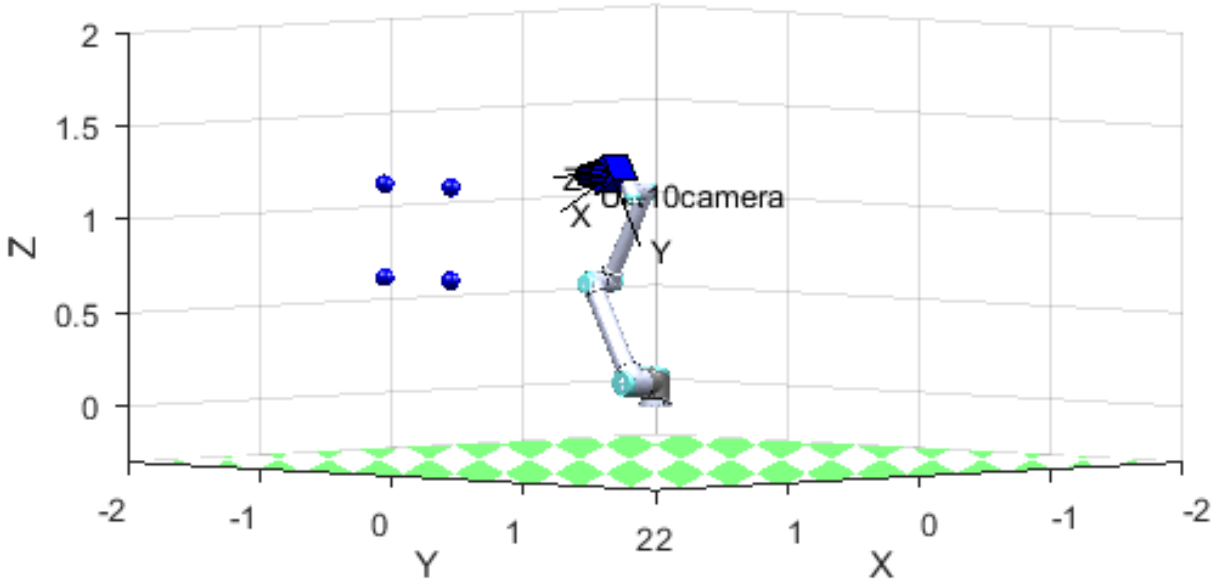


IBVS Remarks

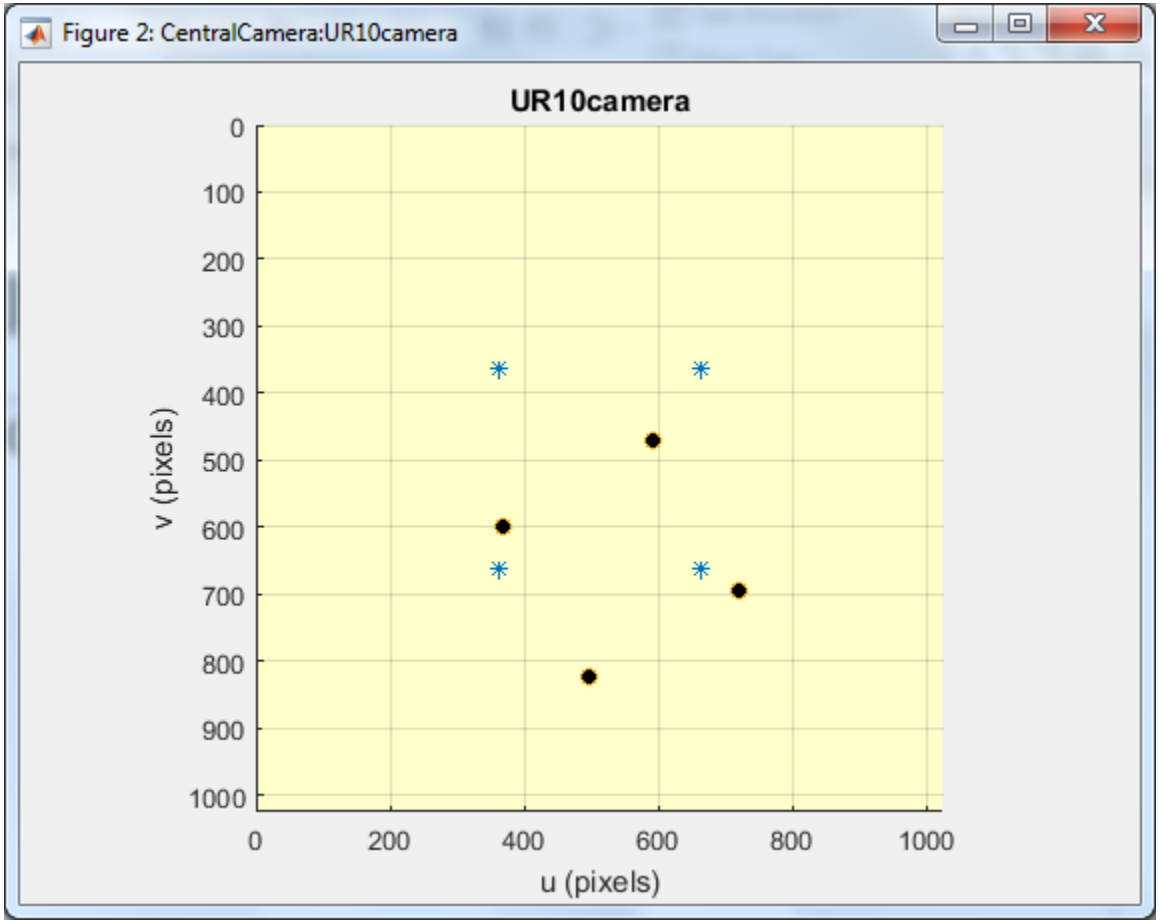
- Positioning accuracy of the system is less sensitive to camera calibration errors
- A depth estimation or approximation is necessary in the design of control law
- The convergence is ensured only in a neighborhood of desired position

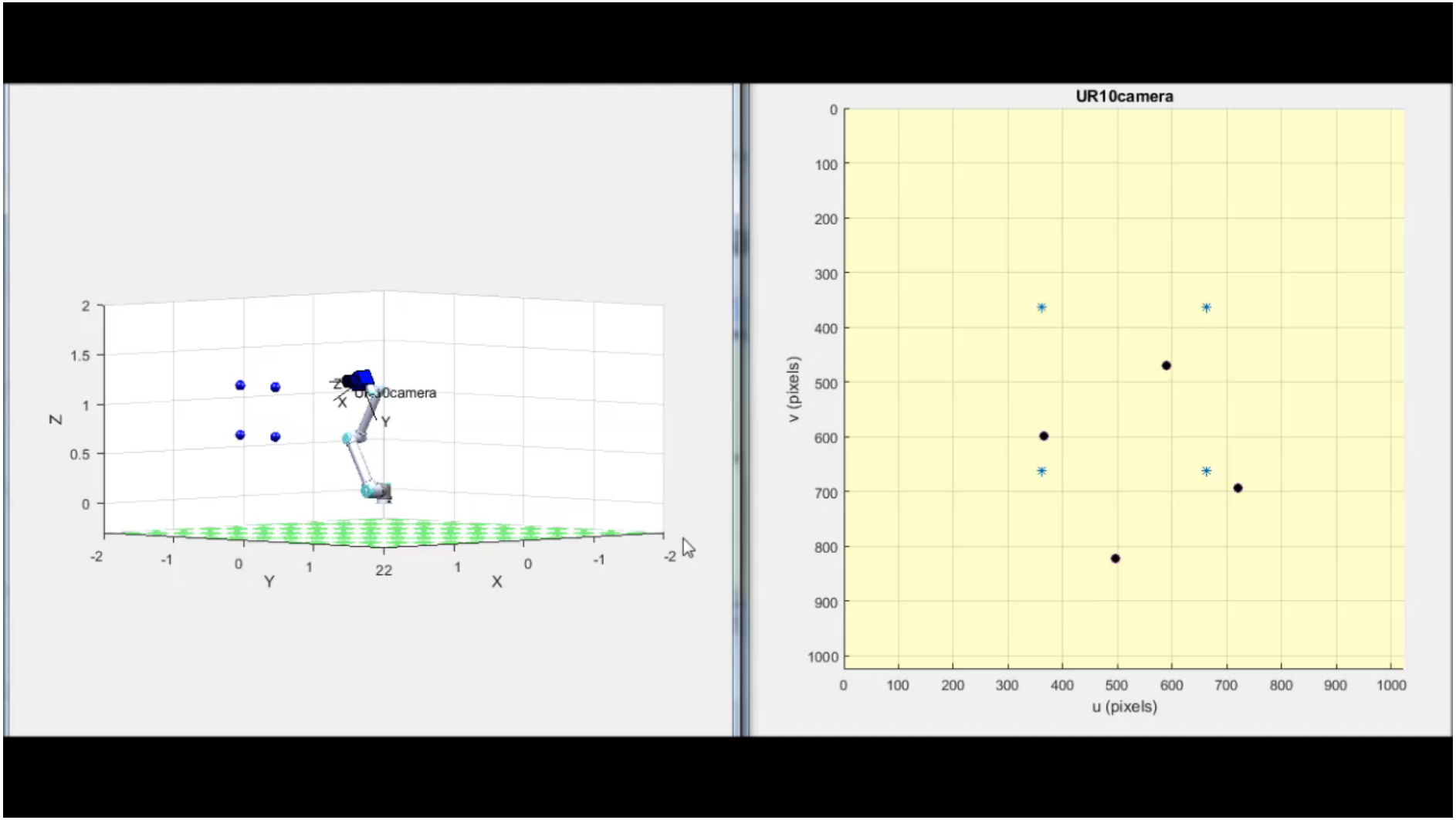


1.1–1.2: Initialise & Plot Model

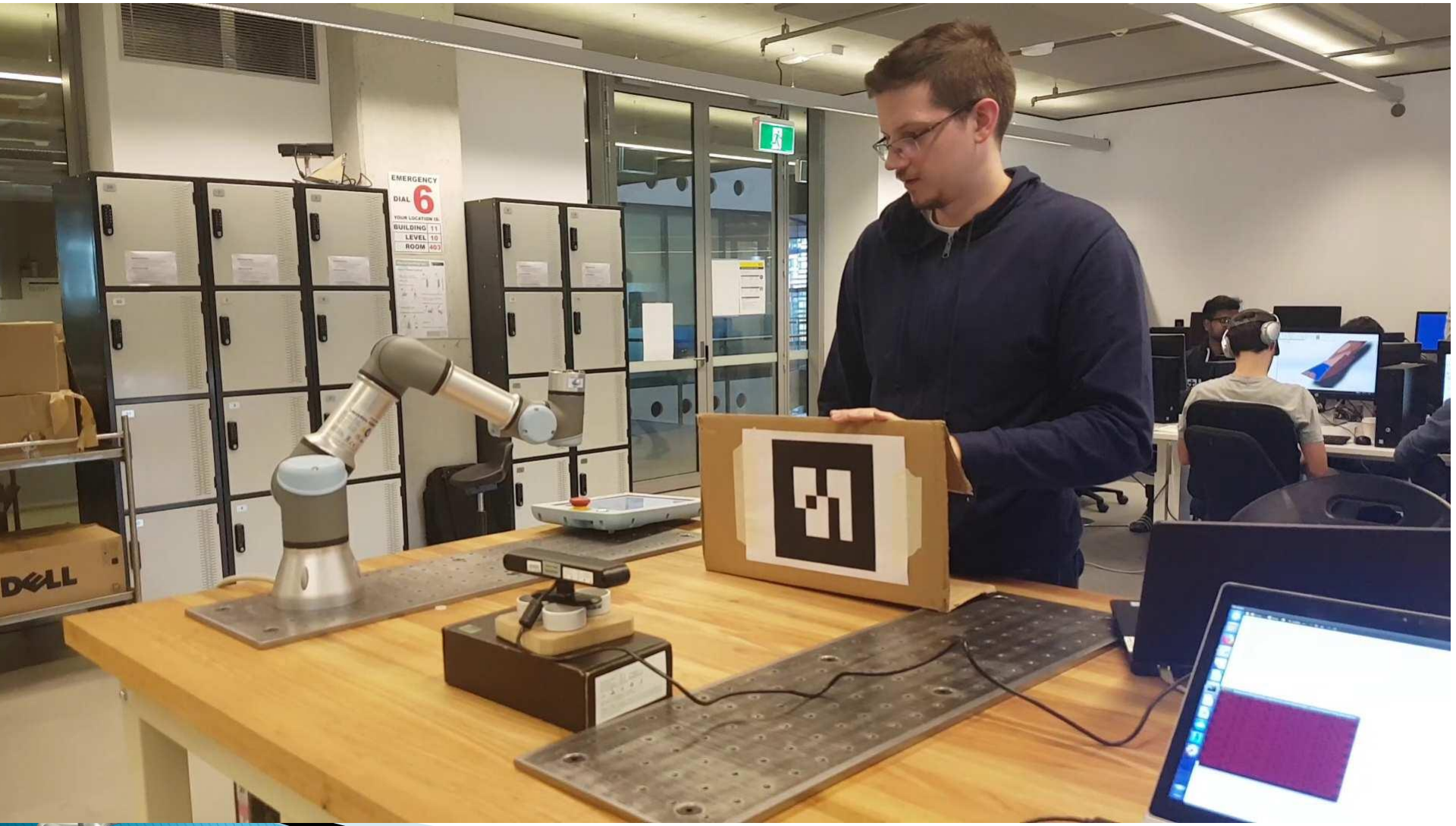


1.3: Start and goal




UR10camera
Z
X
Y
2
1.5
1
0.5
0
-2
-1
0
1
22
1
0
-1
-2
Y
X
UR10camera
v (pixels)
u (pixels)
0
100
200
300
400
500
600
700
800
900
1000

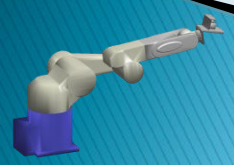
Lab 8: Visual Servoing



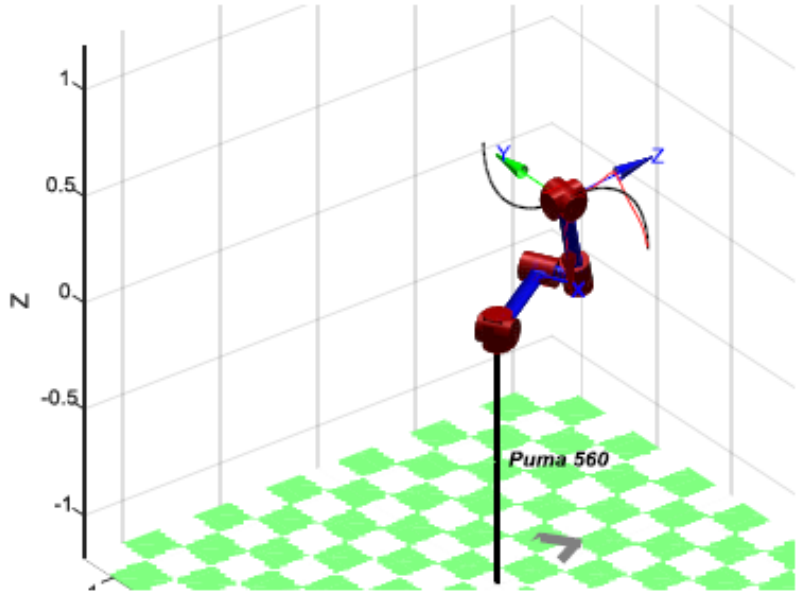
Lab 9 – Tonight and Week 10

- Three questions. Today, focus on Question 1
  - Q1) Resolved Motion Rate Control in 6DOF
  - Q2) Static Torque
  - Q3) Dynamic Torque

- Skeleton code is provided and the lab starter talks through the use (and populating of this skeleton code)



Lab 9 Exercises



1 Resolved Motion Rate Control in 6DOF

(Download **Lab9Question1Skeleton.m** from UTSOnline)

1.1 Set parameters for the simulation

1.2 Allocate array data

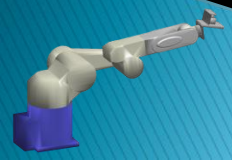
1.3 Set up trajectory, initial pose

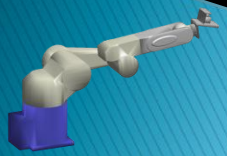
1.4 Track the trajectory with RMRC

1.5 Plot the results

1.6 Consider and discuss the following questions:

- Does the robot successfully track the trajectory? Why or why not?
- Is the robot hitting singularities?
- Are the joints losing control?
- Is Damped Least Squares applied sufficiently?
- Is the trajectory error too big?
- How could you attenuate the damping coefficient to fix this?
- Does the robot hit joint limits?
- How might the initial guess of joint angles when solving inverse kinematics affect this?





Add a reasonable time period and number of steps

Try different weightings for position and angle

Make a Transform for first step. Do inverse kinematics to get the start of the trajectory

For each step use actual and desired transform to compute a required velocity

Note: you may need to go back to RMRC lecture to determine rotational velocities

Get Jacobian and manipulability and check if we should use DLS

Ensure we stay within joint limits

Save and plot result data and simulations

```
% 1.3) Set up trajectory, initial pose
s = lspb(0,1,steps);                % Trapezoidal trajectory scalar
for i=1:steps
    x(1,i) = (1-s(i))*0.35 + s(i)*0.35; % Points in x
    x(2,i) = (1-s(i))*-0.55 + s(i)*0.55; % Points in y
    x(3,i) = 0.5 + 0.2*sin(i*delta); % Points in z
    theta(1,i) = 0;                 % Roll angle
    theta(2,i) = 5*pi/9;            % Pitch angle
    theta(3,i) = 0;                 % Yaw angle
end

% 1.5) Plot the results
figure(1)
plot3(x(1,:),x(2,:),x(3,:),'k','LineWidth',1)
p560.plot(qMatrix,'trail','r-')

for i = 1:6
    figure(2)
    subplot(3,2,i)
    plot(qMatrix(:,i),'k','LineWidth',1)
    title(['Joint ', num2str(i)])
    ylabel('Angle (rad)')
    refline(0,p560.qlim(i,1));
    refline(0,p560.qlim(i,2));

    figure(3)
    subplot(3,2,i)
    plot(qdot(:,i),'k','LineWidth',1)
    title(['Joint ',num2str(i)]);
    ylabel('Velocity (rad/s)')
    refline(0,0)
end

figure(4)
subplot(2,1,1)
plot(positionError'*1000,'LineWidth',1)
refline(0,0)
xlabel('Step')
ylabel('Position Error (mm)')
legend('X-Axis','Y-Axis','Z-Axis')
```

Summary: Redundant Manipulators

- A redundant robot has more joints than task dimensions
  - The Jacobian is not square, and cannot be directly inverted

$$\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{m \times n},\ m < n$$

- There are infinite choices for the joint velocities to perform a given end-effector task using a redundant manipulator
- The *weighted pseudoinverse Jacobian* gives the smallest possible combination of joint velocities to achieve the desired task

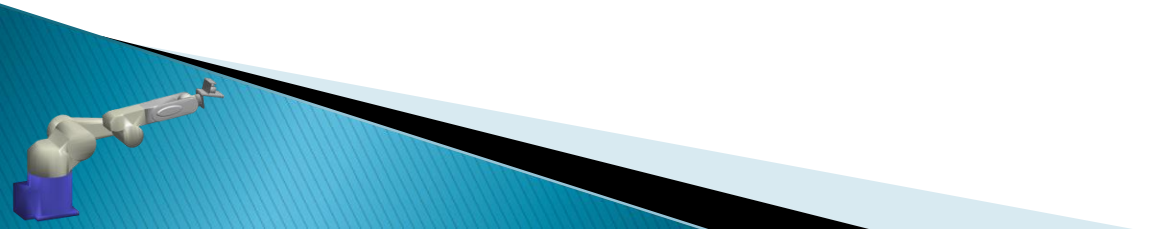
$$\begin{aligned}\dot{\mathbf{q}} &= \mathbf{W}^{-1}\mathbf{J}(\mathbf{q})^{\mathrm{T}}\left(\mathbf{J}(\mathbf{q})\mathbf{W}^{-1}\mathbf{J}(\mathbf{q})^{\mathrm{T}}\right)^{-1}\dot{\mathbf{x}} \\ &= \mathbf{J}_{\mathrm{W}}^{\dagger}(\mathbf{q})\dot{\mathbf{x}}\end{aligned}$$

- The weighted, minimum-velocity solution does not work for a non-redundant robot!

$$\mathbf{J}(\mathbf{q})\mathbf{J}_{\mathrm{W}}^{\dagger}(\mathbf{q})\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})^{-1}\dot{\mathbf{x}}, \qquad \text{if } \mathbf{J}(\mathbf{q}) \in \mathbb{R}^{m \times m}$$

- The weighting matrix can be chosen to avoid joint limits
- Redundant manipulators can perform complex manoeuvres through *null space projection*

$$\dot{\mathbf{q}} = \mathbf{J}^{\dagger}(\mathbf{q})\dot{\mathbf{x}} + \left(\mathbf{I} - \mathbf{J}^{\dagger}(\mathbf{q})\mathbf{J}(\mathbf{q})\right)\mathbf{y}_2$$



Remember this slide from RMRC lecture?
Angular velocities must be derived from the Rotation Matrix
You will need it for Lab 9 Question 1

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}, \mathbf{R} \in \mathbb{SO}(3)$$

$$\mathbf{R}\mathbf{R}^{\mathrm{T}} = \mathbf{I}$$
$$\frac{\mathrm{d}\mathbf{R}}{\mathrm{dt}} = \dot{\mathbf{R}}\mathbf{R}^{\mathrm{T}} + \mathbf{R}\dot{\mathbf{R}}^{\mathrm{T}} = \mathbf{0}$$
$$\dot{\mathbf{R}}\mathbf{R}^{\mathrm{T}} = -\mathbf{R}\dot{\mathbf{R}}^{\mathrm{T}}$$

For simplicity, assume $\mathbf{R} = \mathbf{I}$, then:

$$\dot{\mathbf{R}} = -\dot{\mathbf{R}}^{\mathrm{T}}$$
$$\begin{bmatrix} 0 & -\dot{\varphi} & \dot{\theta} \\ \dot{\varphi} & 0 & -\dot{\phi} \\ -\dot{\theta} & \dot{\phi} & 0 \end{bmatrix} = -\begin{bmatrix} 0 & -\dot{\varphi} & \dot{\theta} \\ \dot{\varphi} & 0 & -\dot{\phi} \\ -\dot{\theta} & \dot{\phi} & 0 \end{bmatrix}^{\mathrm{T}}$$

The roll, pitch, yaw velocities $[\dot{\phi} \quad \dot{\theta} \quad \dot{\varphi}]$ skew symmetric

For the general case:

$$\dot{\mathbf{R}} = \mathrm{S}(\boldsymbol{\omega})\mathbf{R}$$

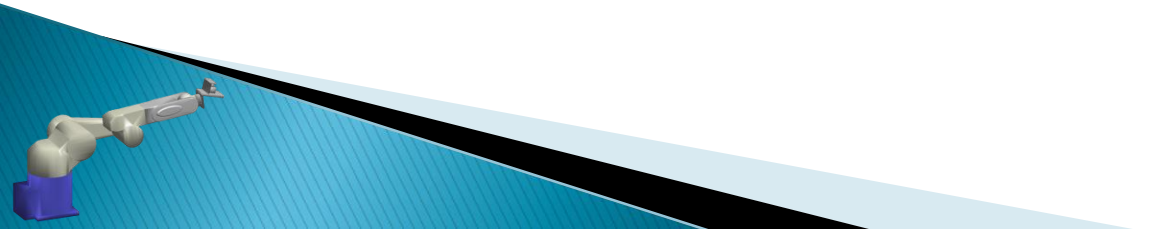
$$\boldsymbol{\omega} = [\dot{\phi} \quad \dot{\theta} \quad \dot{\varphi}]^{\mathrm{T}}$$

$\mathrm{S}(\cdot)$ is the skew-symmetric matrix operator.

$$\mathbf{R}(\mathrm{t}+1) = \mathbf{R}(\mathrm{t}) + \Delta\mathrm{t}\dot{\mathbf{R}}$$
$$\mathrm{S}(\boldsymbol{\omega})\mathbf{R} = \Delta\mathrm{t}^{-1}\big(\mathbf{R}(\mathrm{t}+1) - \mathbf{R}(\mathrm{t})\big)$$
$$\mathrm{S}(\boldsymbol{\omega}) = \Delta\mathrm{t}^{-1}\big(\mathbf{R}(\mathrm{t}+1) - \mathbf{R}(\mathrm{t})\big)\mathbf{R}(\mathrm{t})^{\mathrm{T}}$$
$$= \Delta\mathrm{t}^{-1}\big(\mathbf{R}(\mathrm{t}+1)\mathbf{R}(\mathrm{t})^{\mathrm{T}} - \mathbf{I}\big)$$

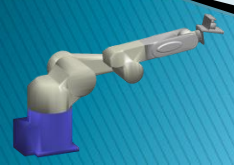
Then extract the angular velocities:

$$\dot{\phi} = \mathrm{S}_{32}$$
$$\dot{\theta} = \mathrm{S}_{13}$$
$$\dot{\varphi} = \mathrm{S}_{21}$$



Summary: Manipulator Statics

- The Jacobian maps a wrench of forces, torques at the end-effector to the joint torques
- Rearranging the energy/work equation:
- $\boldsymbol{\tau} = \mathbf{J}(\mathbf{q})^{\mathrm{T}}\mathbf{w}$
- The reaction torque for a wrench applied at the end-effector is related to the Jacobian.



Summary: Manipulator Dynamics

- Lagrangian mechanics can be used to solve for the dynamic equations

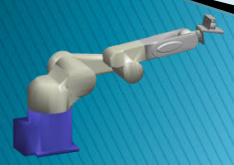
$$\frac{\mathrm{d}}{\mathrm{dt}}\left(\frac{\partial \mathrm{L}}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial \mathrm{L}}{\partial \mathbf{q}} = \boldsymbol{\tau}$$

- $\mathbf{M}(\mathbf{q})$ is the inertia matrix
  - It is symmetric and positive definite
- $\mathbf{c}(\mathbf{q}, \dot{\mathbf{q}})$ is a vector of Coriolis forces
- $\mathbf{g}(\mathbf{q})$ is the gravity vector
- To move between joint angles, solve for the joint accelerations

$$\ddot{\mathbf{q}}(\mathrm{t}) = \Delta \mathrm{t}^{-2}\left(\mathbf{q}(\mathrm{t}+1) - \mathbf{q}(\mathrm{t}) - \Delta \mathrm{t}\dot{\mathbf{q}}(\mathrm{t})\right)$$

- To accelerate the joint, solve the dynamics equation

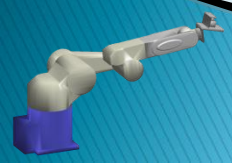
$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q})$$



Note/slides from the textbook

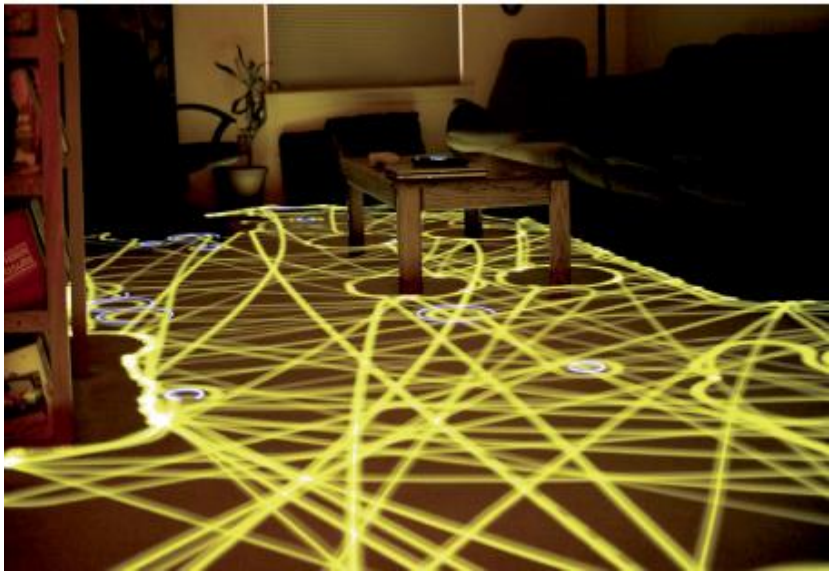
- Textbook readings (Week 9) :
  - Chapter 5 (pages 87-100): “Navigation”
  - Section 8.2 (pages 180-188): “Velocity Relationships” and “Resolved-Rate Motion Control”
  - Section 9.1 (pages 191-200): “Dynamics and Control: Equations of Motion”

- Although it is better to read the textbook, some notes (in slide format) have been summarised below

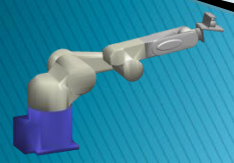


Navigation

- Robot navigation is the problem of guiding a robot towards a goal.
- For example heading towards a light, following a white line on the ground, moving through a maze by following a wall, or vacuuming a room by following a random path.
- The more familiar human-style *map-based navigation* is used by more sophisticated robots.

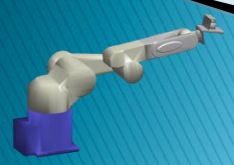


- **Fig. 5.1.** Time lapse photograph of a Roomba robot cleaning a room (photo by Chris Bartlett)



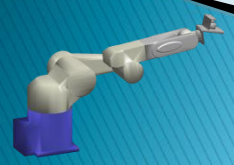
Reactive Navigation

- Insects such as ants and bees gather food and return it to the nest based on input from their senses, they have far too few neurons to create any kind of mental map of the world and plan paths through it.
- a goal oriented machine that can sense, plan and act.
- The manifestation of complex behaviours by simple organisms was of interest to early researchers in cybernetics.



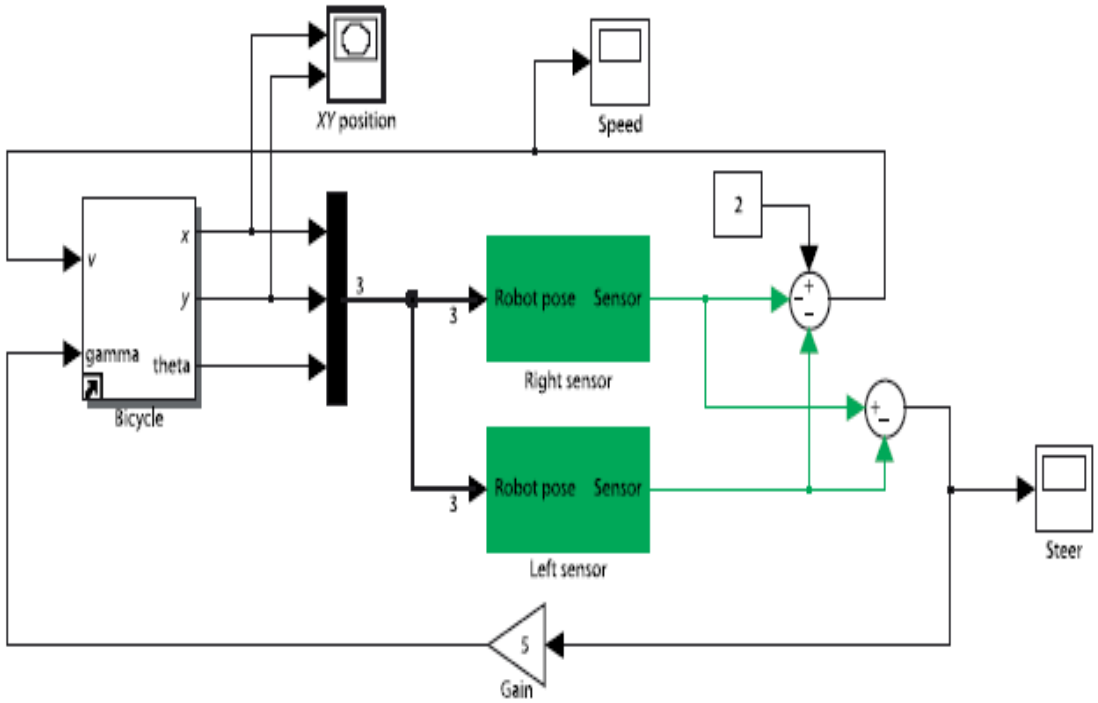
Braitenberg Vehicles

- A very simple class of goal achieving robots are known as Braitenberg vehicles and are characterised by direct connection between sensors and motors.
- Consider the problem of a robot moving in two dimensions that is seeking the maxima of a scalar field – the field could be light intensity or the concentration of some chemical.



Braitenberg Vehicles (continued…)

- **Fig. 5.2.** The Simulink® model sl_braitenberg drives the vehicle toward the maxima of a provided scalar function. The vehicle plus controller is an example of a Braitenberg vehicle

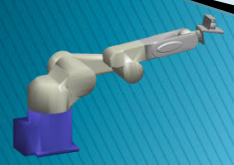


Braitenberg Vehicles (continued…)

- The vehicle speed $v = 2 - s_R - s_L$
  - where s_R and s_L are the right and left sensor readings respectively.
  - At the goal $s_R = s_L = 1$ the velocity becomes zero.
  - Steering angle is based on the difference between the sensor readings:

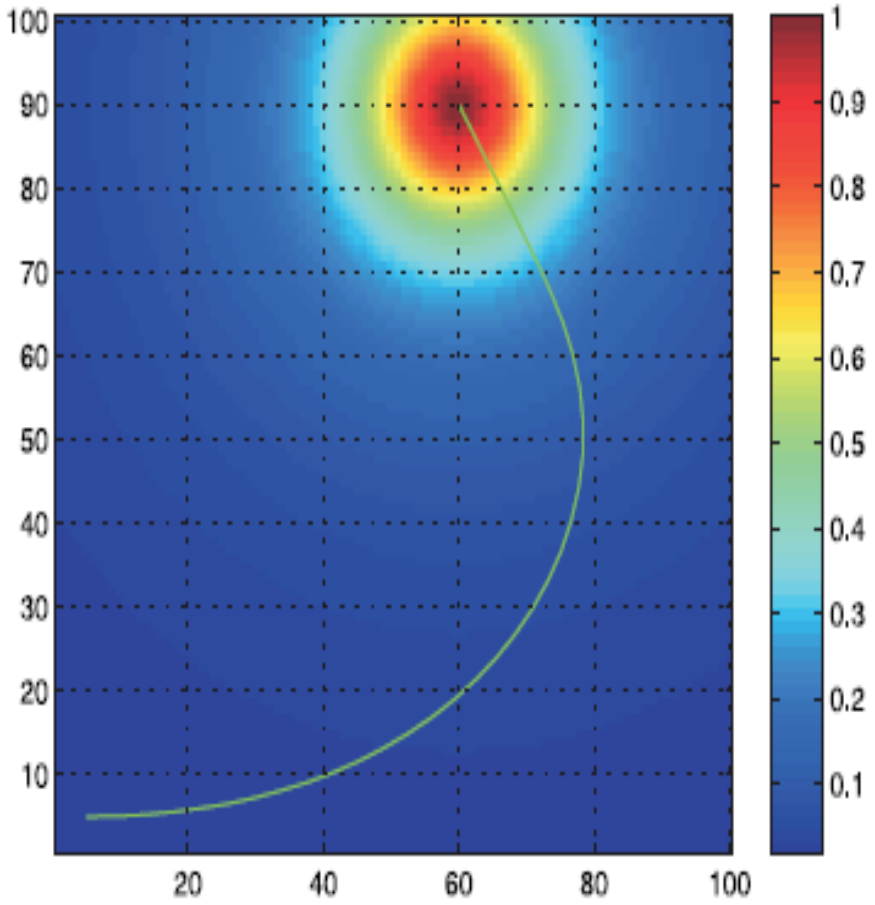
$$\gamma = k(s_R - s_L)$$

  - so when the field is equal in the left- and right-hand sensors the robot moves straight ahead.



Braitenberg Vehicles (continued…)

- **Fig. 5.3.** Path of the Braitenberg vehicle moving toward (and past) the maximum of a 2D scalar field whose magnitude is shown color coded



Simple Automata

- Another class of reactive robots are known as *bugs* – simple automata that perform goal seeking in the presence of non-driveable areas or obstacles.
- There are a large number of *bug* algorithms and they share the ability to sense when they are in proximity to an obstacle.
  - But the *bug* includes a state machine and other logic in between the sensor and the motors.

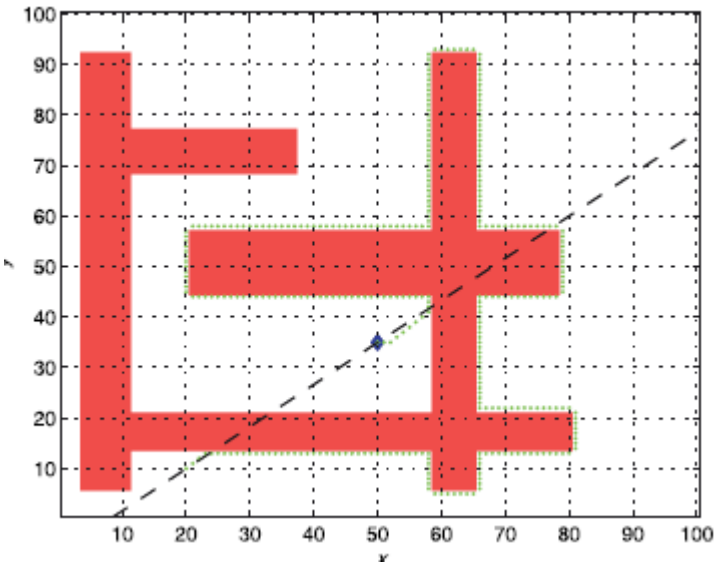
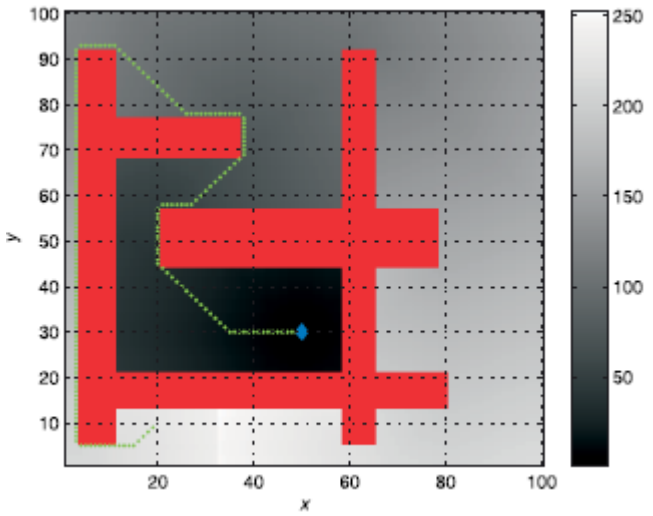


Fig. 5.4. The path taken by the *bug2* algorithm is marked by green dots. The goal is a blue diamond, the black dashed line is the Mline, the direct path from the start to the goal. Obstacles are indicated by red pixels

Map-Based Planning

- There are many ways to represent a map and the position of the vehicle within the map.
  - One approach is to represent the vehicle position as $(x, y) \in \mathbb{R}^2$ and the drivable regions or obstacles as polygons, each comprising lists of vertices or edges.
  - A simpler and computer-friendly representation is the occupancy grid.
  - As its name implies the world is treated as a grid of cells and each cell is marked as occupied or unoccupied.

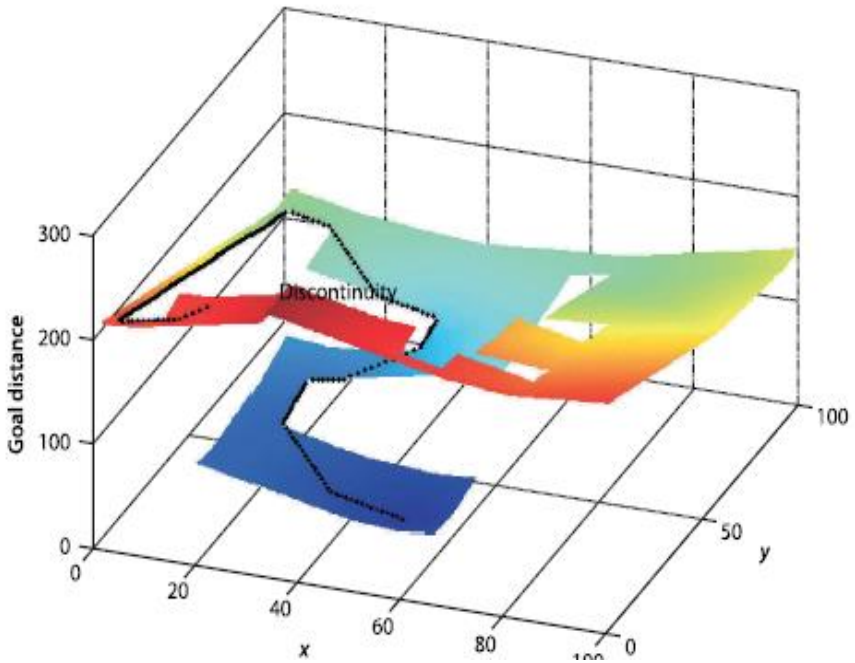
- **Fig. 5.5.** The distance transform path. Obstacles are indicated by red cells. The background grey intensity represents the cell's distance from the goal in units of cell size as indicated by the scale on the right-hand side



Distance Transform

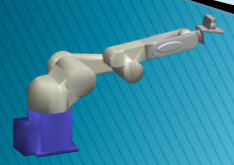
- Consider a matrix of zeros with just a single non-zero element representing the goal.
- The distance transform of this matrix is another matrix, of the same size, but the value of each element is its distance from the original non-zero pixel.

- **Fig. 5.6.** The distance transform as a 3D function, where height is distance from the goal. Navigation is simply a downhill run. Note the discontinuity in the distance transform where the split wavefronts met

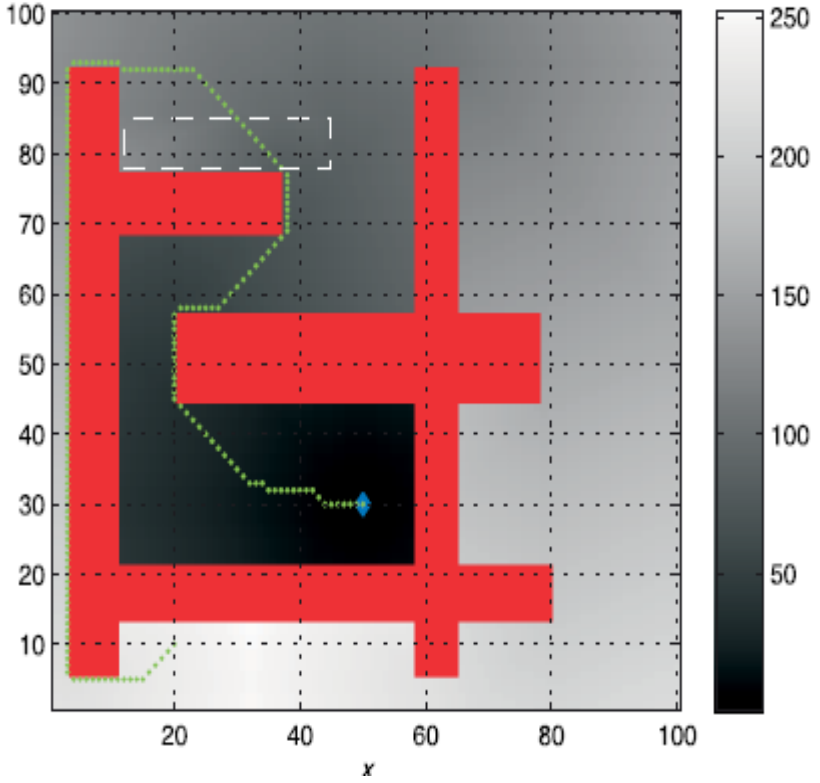


D*

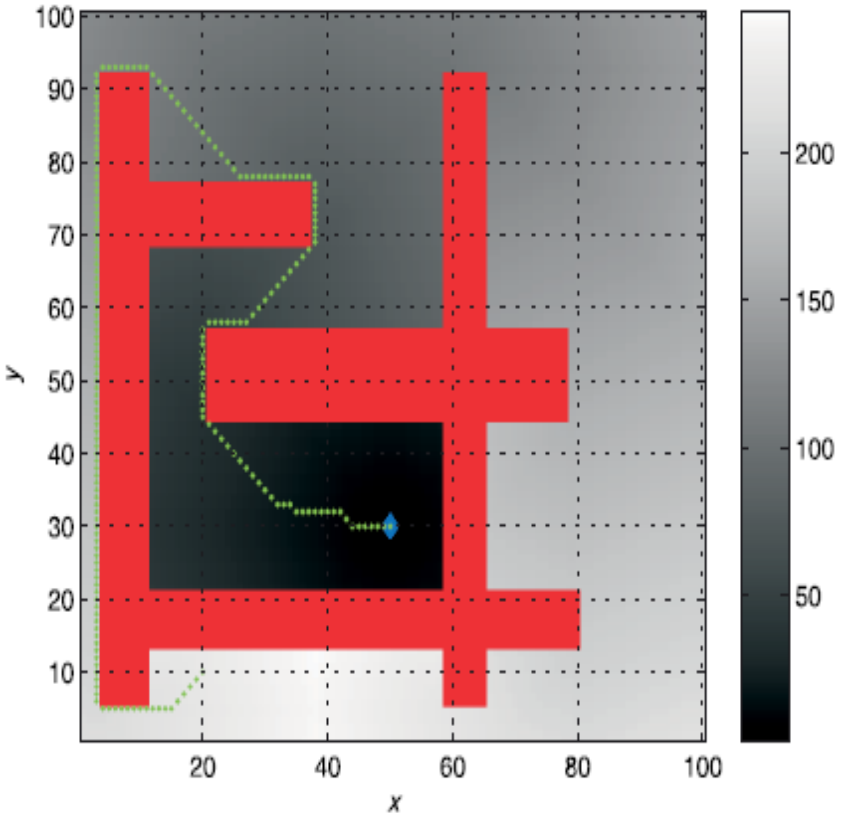
- A popular algorithm for robot path planning is called D*, and it has a number of features that are useful for real-world applications.
- D* finds the path which minimizes the total cost of travel. If we are interested in the shortest time to reach the goal then cost is the time to drive across the cell and is inversely related to traversability.



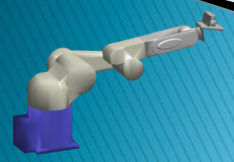
D* (continued…)



- **Fig. 5.7.** The D* planner path. Obstacles are indicated by red cells and all driveable cells have a cost of 1. The background grey intensity represents the cell's distance from the goal in units of cell size as indicated by the scale on the right-hand side

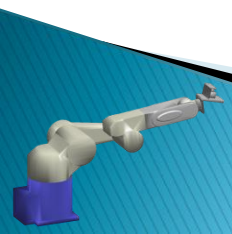


- **Fig. 5.8.** Path from D* planner with modified map. The higher-cost region is indicated by the white dashed rectangle and has changed the path compared to Fig. 5.7



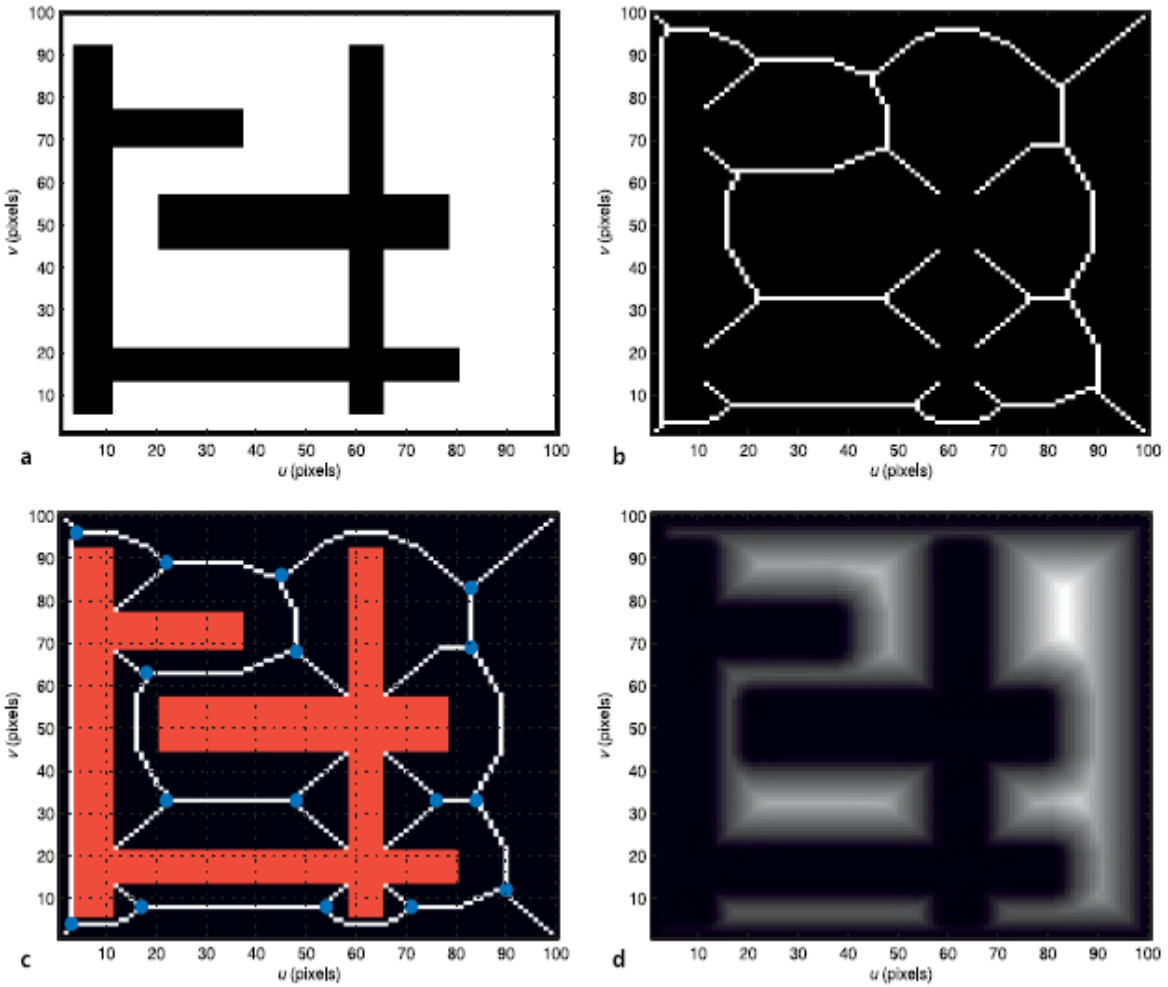
Voronoi Roadmap Method

- In planning terminology the creation of a plan is referred to as the *planning phase*.
- The *query phase* uses the result of the planning phase to find a path from A to B.
- The disparity in planning and query costs has led to the development of roadmap methods where the query can include both the start and goal positions.



Voronoi Roadmap Method (continued…)

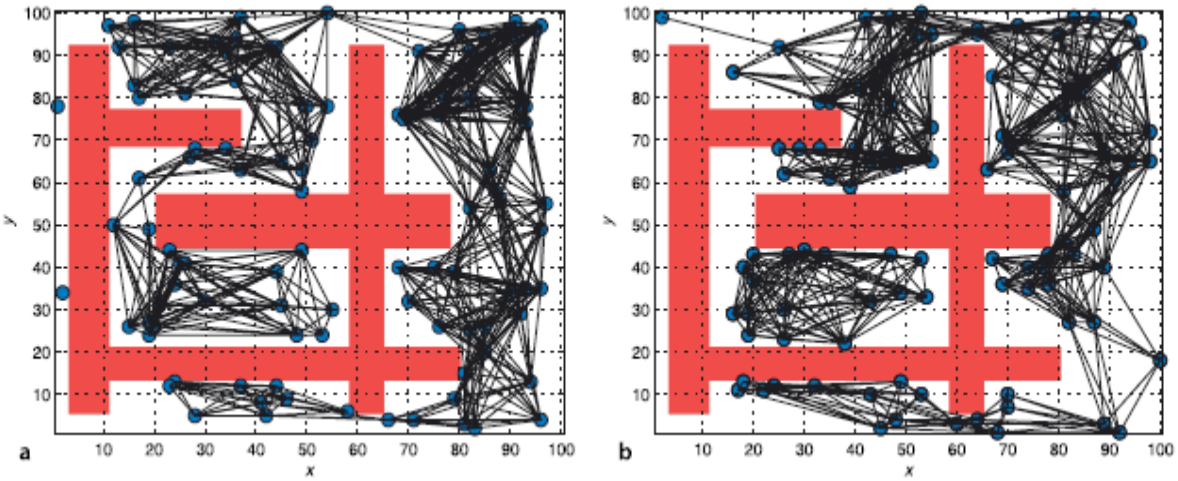
- **Fig. 5.9.** Steps in the creation of a Voronoi roadmap. **a** Free space is indicated by white cells, **b** the skeleton of the free space is a network of adjacent cells no more than one cell thick, **c** the skeleton with the obstacles overlaid in red and roadmap junction points indicated in blue. **d** the distance transform of the obstacles, pixel values correspond to distance to the nearest obstacle



Probabilistic Roadmap Method

- The high computational cost of the distance transform and skeletonization methods makes them infeasible for large maps and has led to the development of probabilistic methods.
- These methods sparsely sample the world map and the most well known of these methods is the probabilistic roadmap or PRM method.

- **Fig. 5.10.** Probablistic roadmap (PRM) planner and the random graphs produced in the planning phase. **a** Almost fully connected graph, apart from two nodes on the left-hand edge, **b** graph with a large disconnected component

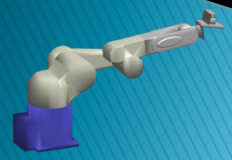


Resolved-Rate Motion Control

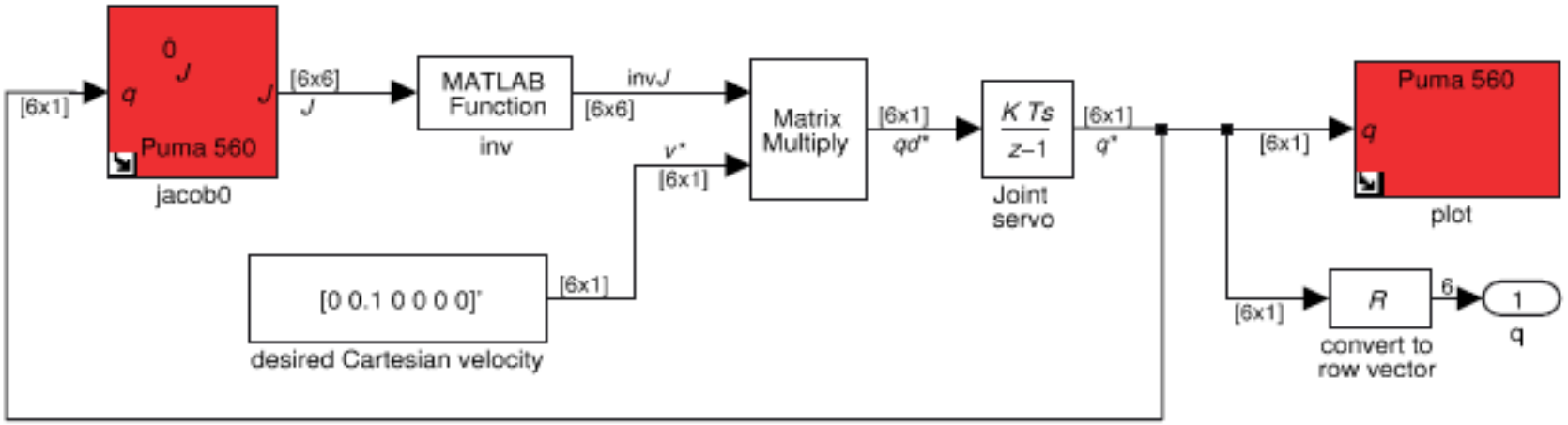
- Resolved-rate motion control: $\dot{q} = J(q)^{-1}v$
  - to map or *resolve* desired Cartesian velocity to joint velocity without requiring inverse kinematics as we used earlier.
  - For now we will assume that the Jacobian is square (6 ×6) and non-singular but we will relax these constraints later.
  - The motion control scheme is typically implemented in discrete-time form as:

$$\dot{q}^*\langle k\rangle = J(q\langle k\rangle)^{-1}v^*$$
$$q^*\langle k+1\rangle = q\langle k\rangle + \delta_t\dot{q}^*\langle k\rangle$$

    - where δ_t is the sample interval. This forward rectangular integration gives the desired joint angles for the next time step, $q^*\langle k+1\rangle$ in terms of the current joint angles and the desired end-effector velocity v^*.



Resolved-Rate Motion Control (continued…)



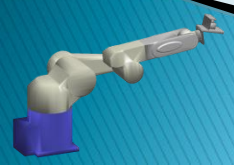
- **Fig. 8.3.** The Simulink® model sl_rrmc for resolved-rate motion control for constant end-effector velocity

Resolved-Rate Motion Control (continued…)

- The approach just described, based purely on integration, suffers from an accumulation of error which we observed as the unwanted *x*- and *z*-direction motion in Fig. 8.4a
- We can eliminate this by changing the algorithm to a *closed-loop* form based on the difference between the desired and actual pose

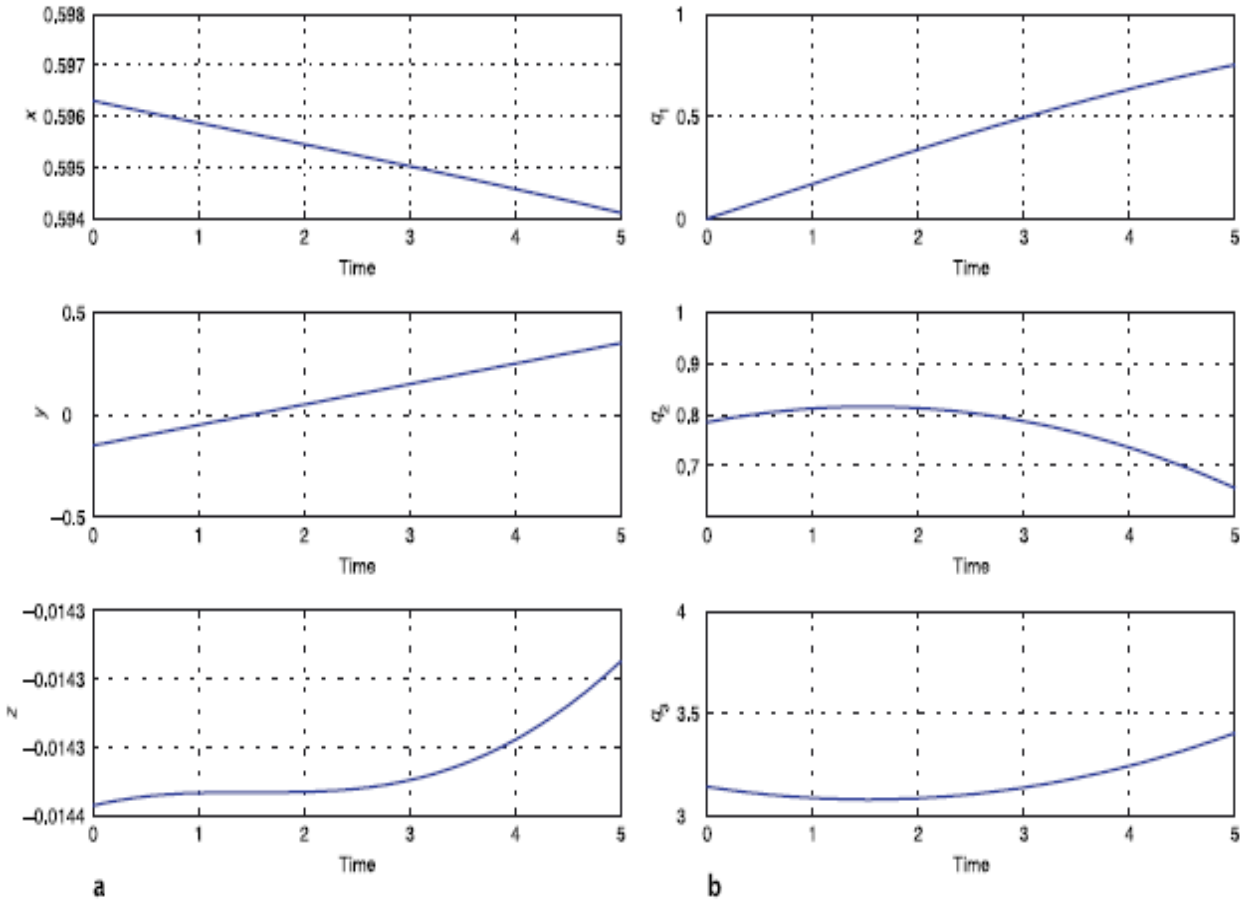
$$\dot{q}^*\langle k\rangle = J(q\langle k\rangle)^{-1}\left(\xi^*\langle k\rangle \ominus \mathcal{K}(q\langle k\rangle)\right)$$
$$\dot{q}^*\langle k+1\rangle = q\langle k\rangle + K_p\delta_t\dot{q}^*\langle k\rangle$$

  - where K_p is a proportional gain.
  - The input is now the desired pose $\xi^*\langle k\rangle$ as a function of time rather than v^*.

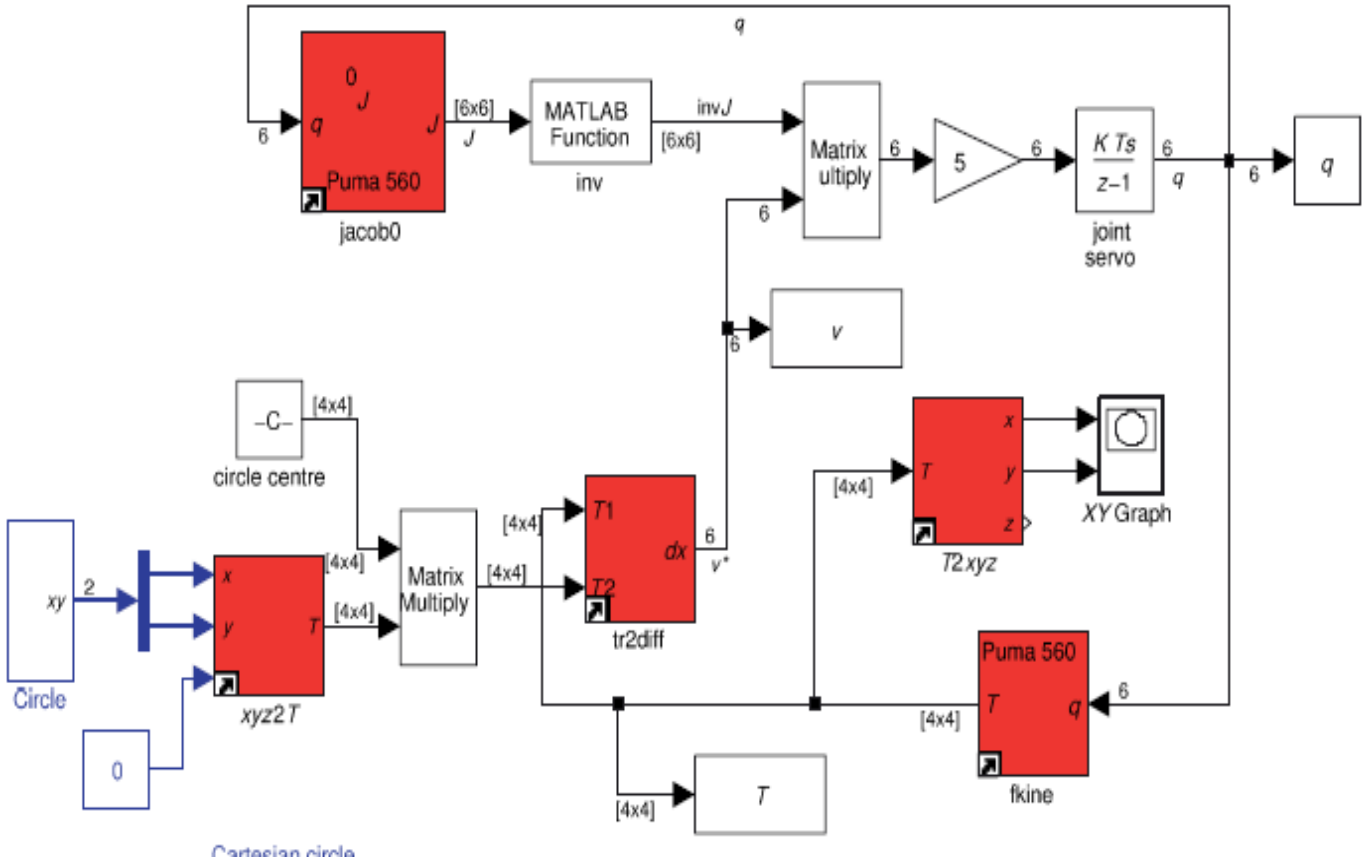


Resolved-Rate Motion Control (continued…)

- **Fig. 8.4.** Resolved-rate motion control, Cartesian and joint coordinates versus time.
  a Cartesian end-effector position;
  b joint coordinates

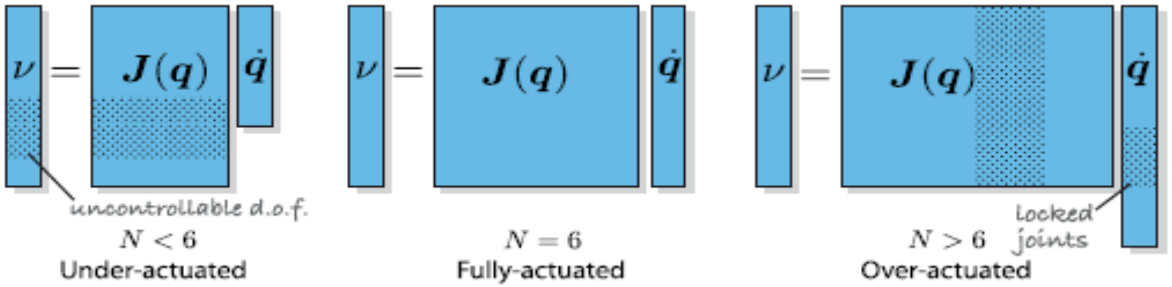


Resolved-Rate Motion Control (continued…)



- **Fig. 8.5.** The Simulink® model sl_rrmc2 for closed-loop resolved-rate motion control with circular end-effector motion

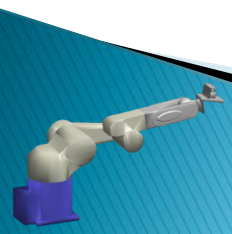
Jacobian Singularity



- **Fig. 8.6.** Schematic of Jacobian, *v* and ¸ for different cases of *N*. The dotted areas represent matrix regions that could be deleted in order to create a square subsystem capable of solution

Jacobian Singularity (continued…)

- The pseudo-inverse of the Jacobian J^+ has the property that $J^+\mathrm{J}=1$
  - just as the inverse does, and is defined as $J^+ = (J^T J)^{-1} J^T$
  - The solution: $q = \dot{J(q)}^+ v$ provides a least squares solution for which$|J_{\dot{q}} - v|$ is the smallest.

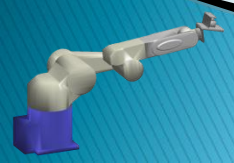


Jacobian for Under-Actuated Robot

- We have to confront the reality that we have *only* two degrees of freedom which we will use to control just v_x and v_y.

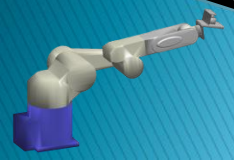
$$\begin{bmatrix} v_x \\ v_y \\ \hline v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} J_{xy} \\ J_0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}$$

  - and taking the top partition, the first two rows, we write $\begin{bmatrix} v_x \\ v_y \end{bmatrix} = J_{xy} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}$
  - where J_{xy} is a 2×2 matrix.
  - we invert this $\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} = J_{xy}^{-1} \begin{bmatrix} v_x \\ v_y \end{bmatrix}$



Jacobian for Over-Actuated Robot

- An over-actuated or redundant robot has $N > 6$, and a Jacobian that is wider than it is tall. In this case we rewrite Eq. 8.6 to use the left pseudo-inverse $\dot{q} = J(q)^{+}v$
  - which, of the infinite number of solutions possible, will yield the one for which $|\dot{q}|$ is smallest – the minimum-norm solution.
  - This is remarkably useful because it allows Eq. 8.9 to be written as $\dot{q} = \underbrace{J(q)^{+}v}_{end-effector\ motion} + \underbrace{NN^{+}\dot{q}_{ns}}_{null-space\ motion}$
  - where the $N \times N$ matrix NN^{+} *projects* the desired joint motion into the null-space so that it will not affect the end-effector Cartesian motion, allowing the two motions to be superimposed.

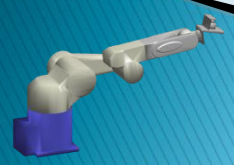


Force Relationships

- concept of a spatial velocity:

$v = (v_x, v_y, v_z, \omega_x, \omega_y, \omega_z)$

- For forces there is a spatial equivalent called a wrench $g = (f_x, f_y, f_z, m_x, m_y, m_z) \epsilon \mathbb{R}^6$ which is a vector of forces and moments.

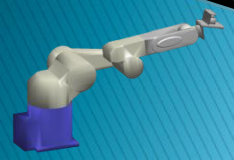


Transforming Wrenches between Frames

- It can be used to map wrenches between coordinate frames. For the case of two frames attached to *the same* rigid body

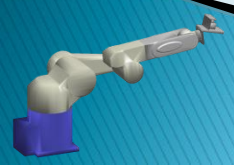
$$A_g = \left(B_{J_A}\right)^T B_g$$

  - where B_{J_A} is given by either Eq. 8.4 or 8.5 and is a function of the relative pose A_{T_B} frame {*A*} to frame {*B*}.
  - Note that the force transform differs from the velocity transform in using the transpose of the Jacobian and the mapping is reversed – it is from frame {*B*} to frame {*A*}.



Transforming Wrenches to Joint Space

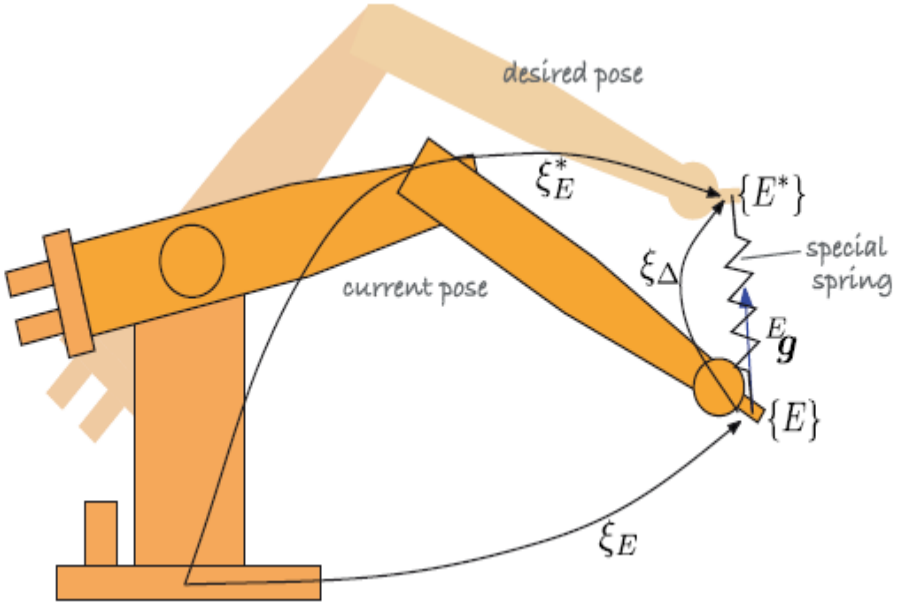
- If the wrench is defined in the end-effector coordinate frame then we use instead $Q = N_{J(q)^T} N_g$
- Interestingly this mapping from external quantities (the wrench) to joint quantities (the generalized forces) can never be singular as it can be for velocity.



Inverse Kinematics: a General Numerical Approach

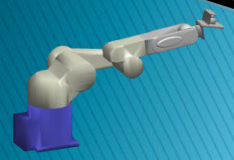
- The principle is shown in Fig. 8.7. The virtual robot is drawn solidly in its current pose and faintly in the desired pose. From the overlaid pose graph we write $\xi_E^* = \xi_E \oplus \xi_\Delta$
  - which we can rearrange as $\xi_\Delta = \ominus \xi_E \oplus \xi_E$

- **Fig. 8.7.** Schematic of the Numerical inverse kinematic approach, showing the current ξ_E and the desired ξ_E^* manipulator pose



Inverse Kinematics: a General Numerical Approach (continued…)

- We postulate a *special* spring between the end-effector of the two poses which is pulling (and twisting) the robot's end-effector toward the desired pose with a wrench proportional to the *difference* in pose $E_g \alpha \Delta(\xi_E, \xi_E^*)$
  - The wrench is also a 6-vector and comprises forces and moments. We write $E_g = \Upsilon^{\Delta}(\xi_E, \xi_E^*)$
  - where Υ is a constant and the current pose is computed using forward kinematics $\xi_E\langle k \rangle = \mathcal{K}(q\langle k \rangle)$
  - where $q\langle k \rangle$ is the current estimate of the inverse kinematic solution.
  - The end-effector wrench Eq. 8.14 is *resolved* to joint forces: $Q\langle k \rangle = E_{Jq\langle k \rangle^T} E_{g\langle k \rangle}$

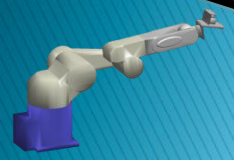


Equations of Motion

- A very efficient way for computing

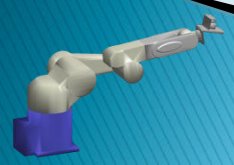
$$Q = M(q)\dot{q} + C(q,\dot{q})\dot{q} + F(\dot{q}) + G(q) + J(q)^T g$$

  - is the recursive Newton-Euler algorithm which starts at the base and working outward adds the velocity and acceleration
  - where q, $\dot{q}$, and $\ddot{q}$ are respectively the vector of generalized joint coordinates, velocities
  - and accelerations, ***M*** is the joint-space inertia matrix, ***C*** is the Coriolis and centripetal
  - coupling matrix, ***F*** is the friction force, ***G*** is the gravity loading, and Q is the vector of generalized actuator forces associated with the generalized coordinates q.
  - The last term gives the joint forces due to a wrench g applied at the end effector and ***J*** is the manipulator Jacobian.
  - This equation describes the manipulator rigid-body dynamics and is
  - known as the inverse dynamics – given the pose, velocity and acceleration it computes the required joint forces or torques.

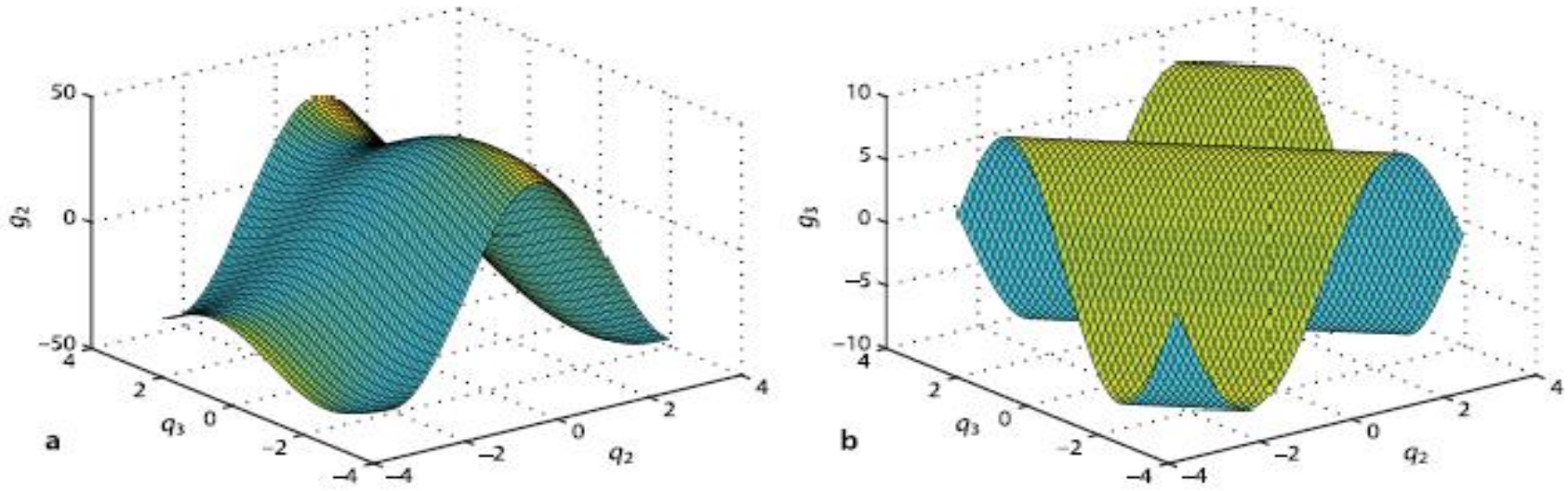


Gravity Term

- $Q = M(q)\dot{q} + C(q,\dot{q})\dot{q} + F(\dot{q}) + G(q) + J(q)^T f$
- We start our detailed discussion with the gravity term because it is generally the dominant term in Eq. 9.1 and is present even when the robot is stationary or moving slowly.
- Some robots use counterbalance weights or even springs to reduce the *gravity* torque that needs to be provided by the motors – this allows the motors to be smaller and thus lower in cost.



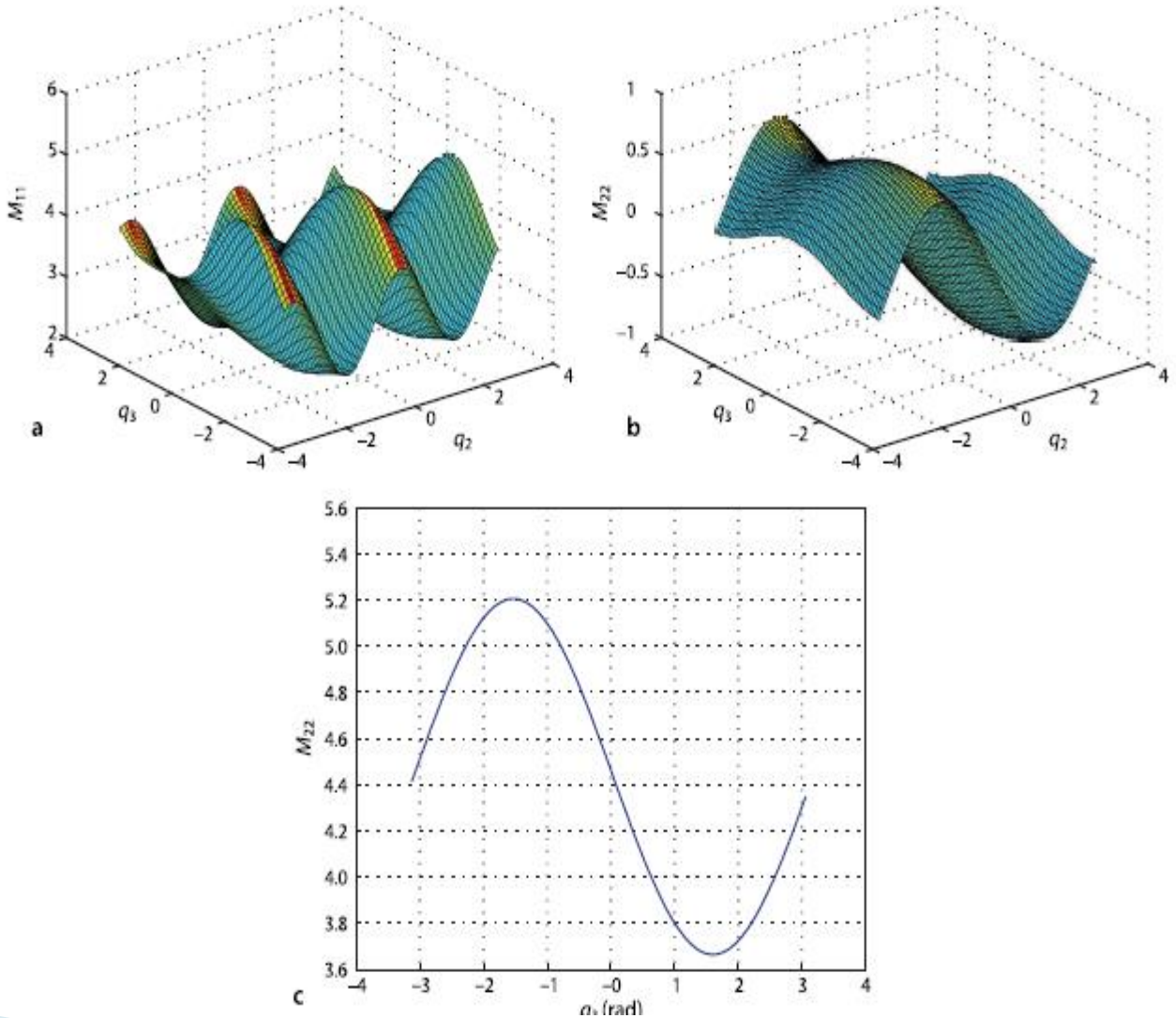
Gravity Term (continued…)



- **Fig. 9.1.** Gravity load variation with manipulator pose.
- **a** Shoulder gravity load,$g_2(q_2, q_3)$;
- **b** elbow gravity load $g_2(q_2, q_3)$

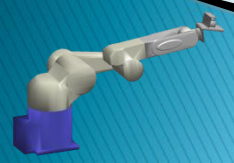
Inertia Matrix

- **Fig. 9.2.** Variation of inerti matrix elements as a function of manipulator pose. **a)** Joint 1 inertia as a function of joint 2 and 3 angles $\boldsymbol{M_{11}}(q_2, q_3)$;**b)** product of inertia $\boldsymbol{M_{12}}(q_2, q_3)$; **c)** joint 2 inertia as a function of joint 3 angle $\boldsymbol{M_{22}}(q_3)$. Inertia has the units of kg m2



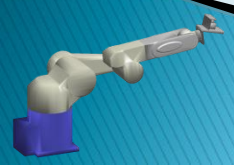
Coriolis Matrix

- $Q = M(q)\dot{q} + C(q,\dot{q})\dot{q} + F(\dot{q}) + G(q) + J(q)^T f$
- The Coriolis matrix $\boldsymbol{C}$ is a function of joint coordinates and joint velocity.
- The centripetal torques are proportional to $\dot{q}_i^2$, while the Coriolis torques are proportional to $\dot{q}_i\dot{q}_j$.
- For example, at the nominal pose with all joints moving at 0.5 rad s^{-1}



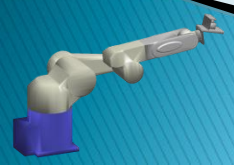
Effect of Payload

- Any real robot has a specified maximum payload which is dictated by two dynamic effects.
  - The first is that a mass at the end of the robot will increase the inertia *seen* by the joints which reduces acceleration and dynamic performance.
  - The second is that mass generates a weight force which the joints needs to support. In the worst case the increased gravity torque component might exceed the rating of one or more motors.
  - However even if the rating is not exceeded there is less torque available for acceleration which again reduces dynamic performance.
    - As an example we will add a 2.5 kg point mass to the Puma 560 which is its rated maximum payload.
    - The centre of mass of the payload cannot be at the centre of the wrist coordinate frame, that is inside the wrist, so we will offset it 100 mm in the z-direction of the wrist frame.



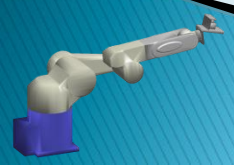
Base Force

- A moving robot exerts a wrench on its base, a vertical force to hold it up and other forces and torques as the arm moves around.



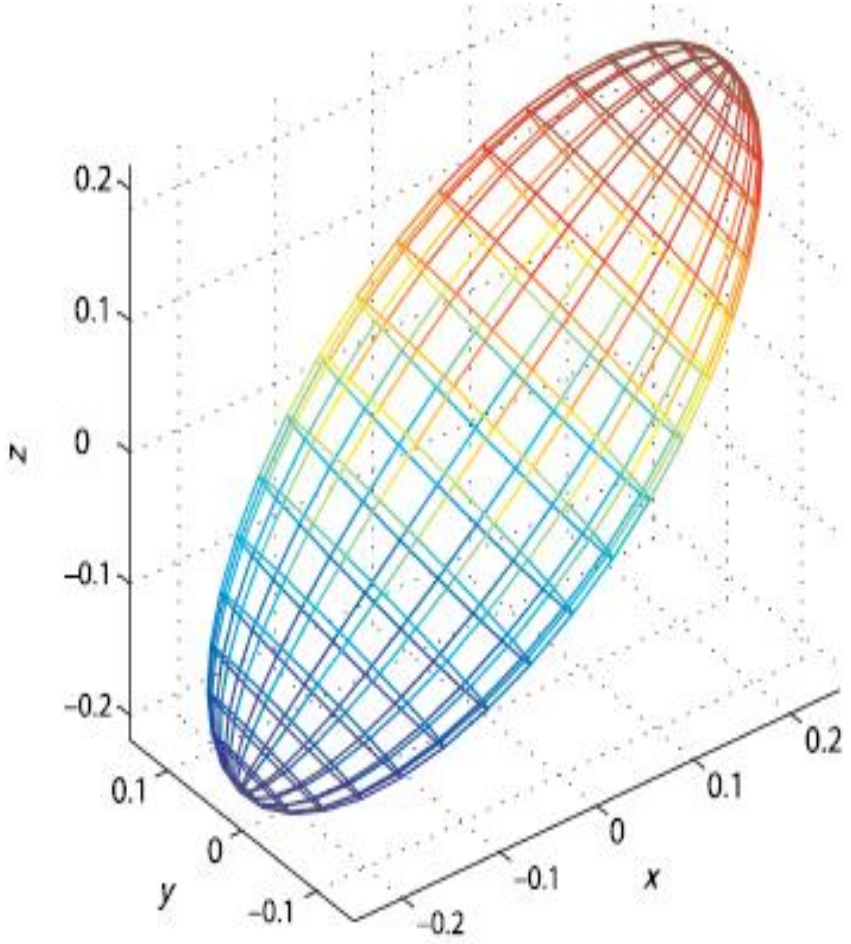
Dynamic Manipulability

- An extension of that measure is to consider how well the manipulator is able to accelerate in different Cartesian directions.
  - Following a similar approach, we consider the set of generalized joint forces with unit norm $Q^T Q = 1$
  - From Eq. 9.1 and ignoring gravity and assuming $\dot{q}=0$ we write: $Q = M\ddot{q}$
  - Differentiating Eq. 8.2 and still assuming $\dot{q}=0$ we write: $\dot{v} = J(q)\ddot{q}$
  - Combining these we write: $\dot{v}^T (JM^{-1}M^T J^T)^{-1}\dot{v} = 1$
  - or more compactly: $\dot{v}^T M_x^{-1}\dot{v} = 1$



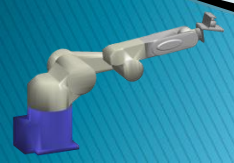
Dynamic Manipulability (continued…)

- **Fig. 9.3.** Spatial acceleration ellipsoid for Puma 560 robot in nominal pose



Dynamic Manipulability (continued…)

- The manipulability measure proposed by Asada is similar but considers the ratios of the eigenvalues of $\ddot{x}^T J^{-1} M J^{-1} \ddot{x} = 1$
- and returns a uniformity measure $m \in [0, 1]$ where 1 indicates uniformity of acceleration in all directions.



References

- Corke PI (2007) A simple and systematic approach to assigning Denavit–Hartenberg parameters. IEEE T Robotic Autom 23(3):590–594