TACTICAL UNMANNED GROUND VEHICLE FOR CLOSE-QUARTERS SURVEILLANCE & COMBAT

MACHINE DESIGN

Aman Malekar (180100012)
Barath Krishna S. (180100027)
Kritti Sharma (180100060)
Mitalee R. Oza (180100067)
Shubham Agrawal (180041000)
Rishab Khantwal (180100095)
Kratik Bhadoriya(180100059)

1 Controls and Dynamics

1.1 Gait Patterns

Like most biological creatures walk by alternatively lifting and placing the legs from and on the ground, in legged locomotion we go with a very similar idea. The entire motion is cyclic in nature with the same motion being repeated for each leg with a phase difference. To understand more of this, we first focus on movement of a single leg over a cycle.
The motion of a leg can be divided into two parts namely, swing phase and stance phase. In the swing phase the foot is not in contact with the ground and is moving fast in the direction of motion of the bot to place a foothold at the next location which is further in the direction of motion. In the stance phase, the foot remains in contact with the ground and provides the force necessary to push the bot forward and balance its weight.

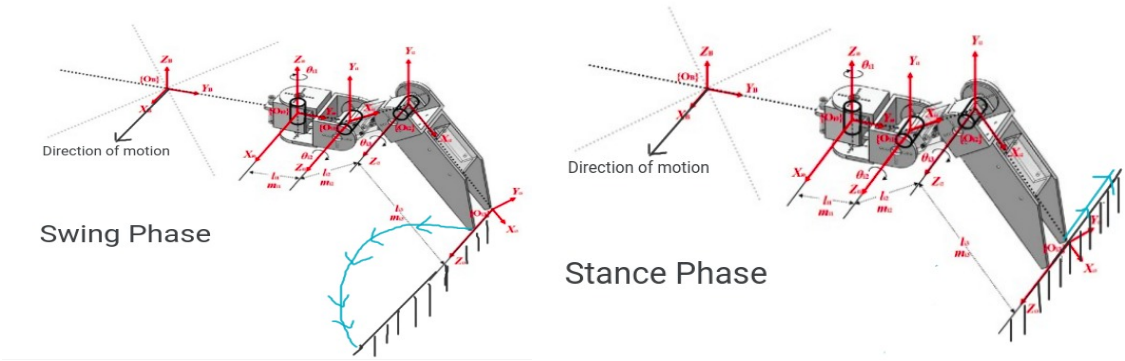


Figure 1: The motion of foot wrt to base in swing and stance phase

The above figure shows the motion of the foot from the frame of reference of the base of the bot. During swing the foot moves faster than the base in the direction of motion of bot to place foot while in stance as the foot contact is stationary wrt to the ground it appears to be moving backward wrt to the base. If we combine these two motions we get the motion of legs over each cycle. The curve defining this motion is generated using CPG which has been discussed in detail in section 1.2 .

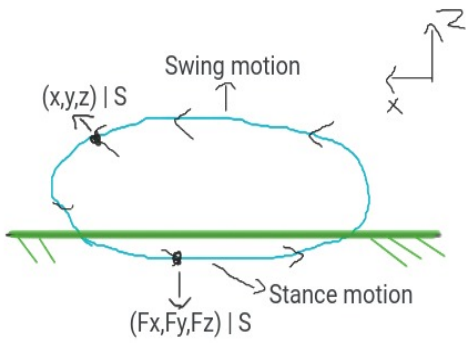


Figure 2: Cyclic motion of foot

The motion of the foot can be summarised as shown in figure above. In the swing phase we specify the position of the foot while in stance we specify the force which the foot should apply to the ground. We can consider a parameter S which goes from one point to another on this curve as S goes from 0 to 1. This parameter is called phase.Phase for each leg Si along with the characteristic of the curve wrt time solves the locomotion problem.

In legged locomotion the driving force and stabilizing force depends on the location and the forces being applied at the point of contact of feet with the ground, which means force will act at the location of foot of legs in stance phase. Hence determining the relative phase difference between the legs is critical to the static and dynamic stability of the bot. For a flat terrain defining the relative phase defines a different gait/walking pattern. On rough terrain where touchdown can happen before or after anticipated due to rocks or potholes, these phases are determined in a closed feedback loop by gait pattern modulator.

To define the phase difference in case of flat terrain, let's discuss the time taken for stance and swing. Let the time for swing phase be Ts and stance phase be Tst. As swing phase is all about placing the leg at another location while leg is in air it doesn't affect the overall speed of the bot and is only limited by actuators and leg dynamics. Hence it is fixed for a given bot. While in stance phase we have the leg in contact with ground while the base is moving. Hence the speed of the bot is Lstride/Tst (see fig2), which means Tst is dependent on speed and reduces for increasing speed. But if Tst becomes very small it would mean that the legs would spend more time in swing which would mean a lesser number of legs would be in touch with ground and stability as well as payload capacity would be affected. The number of legs touching the ground at any instant is also affected by the phase difference between the legs. Hence the selection of gait pattern along with walking speed is critical to stability and has to be optimised. The daig below shows the relative phases between the legs of the hexapods for some standard gait patterns.

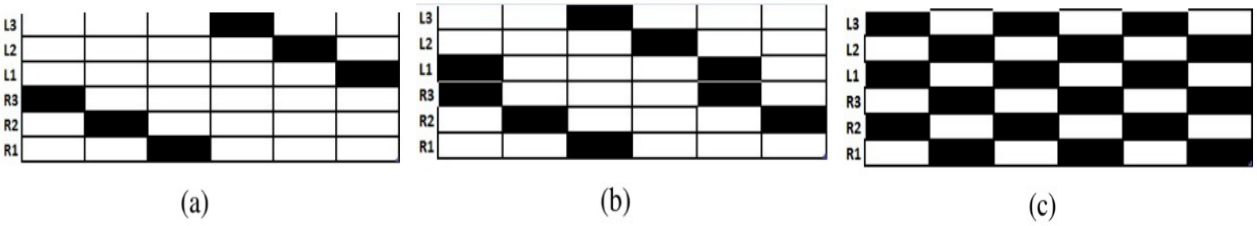


Figure 3: Gait diagram depicting event sequences for three different hexapodal gaits. White color indicates that the foot is in ground contact. a) Metachronal (low - speed) Gait. b) Ripple (medium - speed) Gait. c) Tripod (fast - speed) Gait.

1.2 Central Pattern Generator

Among the various papers we referred to for legged locomotion, hexapod locomotion in particular, the common technique used was that of Central Pattern Generators (CPG). CPG.s are in fact borrowed from Neuroscience for robotic applications.
It is central to the survival of animals to be able to move across complex terrains and environments in search for food, to avoid predators and search for mates. Hence, many aspects

of their morphology and central nervous systems are shaped by the need for efficient locomotion. Similar need in modern day robots has led to the intersection of robotics and neuroscience. Robotics takes inspiration from biology in terms of morphologies, modes of locomotion and/or control mechanisms. The dependence has also lately been the other way, where robots are being used to test biological hypothesis.
The exact way in which the brains work for locomotion is highly complex and there is still ongoing research in this field. What we know is that evolution has developed efficient ways to abstract very complex locomotion patterns, and turn them into simple control signals that can be effortlessly handled by the brain. What we know is that evolution has developed efficient ways to abstract very complex locomotion patterns, and turn them into simple control signals that can be effortlessly handled by the brain. Locomotion is a reasonably periodic activity. Based of the research in this field, it is reasonable to assume that neural circuits do not explicitly control the position of every articulation, but rather modulate higher level parameters such as amplitude and frequency of each gait. This is primarily what Central Pattern Generators in robotic systems exploit.
Depending on the dynamics of the mechanism, central pattern generators produce gait patterns such that the joint angles follow limit cycles, hence producing repetitive motion. The limit cycle is defined by several predefined parameters. The ability to walk on unstructured terrain comes by 'intelligently' changing the parameters of these limit cycles.These parameters are changed based on the sensor inputs from the surroundings such as LIDAR readings, visual inputs etc. The challenge here is to suitably alter parameters based on real time sensor data. Various techniques are used for this purpose, such as Reinforcement Learning, Neural Nets etc. Central Pattern Generators have been used for replicating the locomotion of various vertebrates in robotic systems.

1.3 Gait and Trajectory Optimization

In legged systems, the base movement cannot be independently determined and is dependent on the contact of the feet with ground. Therefore, the heart of the problem is to optimize these contact forces to achieve a desired motion, while also taking into account various restrictions. The ultimate goal is to transcribe this continuous-time optimization problem into one with finite number of decision variables and constraints to be solvable by a Nonlinear Programming Problem (NLP). To this end, multiple physical models have been proposed:

- **Dynamic Models:** Kalakrishnan et al. [2010] presented a general quadruped locomotion controller where the controller includes a procedure that learns optimal foothold choices using terrain templates, a body trajectory optimizer based on the Zero Moment Point (ZMP) stability criterion, and a floating-base inverse dynamics controller that does not require knowledge of the contact forces. The foot planner proposed in Winkler et al. [2014] searches for specific footholds that roughly guide the robot along the pre-planned body path. The nominal footholds corresponding to a virtual Center of Gravity (CoG) position on the planned body path are calculated. Pardo et al. [2017] use a (6+n)-dimensional full rigid-body dynamics considering the mass and inertia of each link and defines the relation between joint torques and base- and joint-accelerations.

As discussed in Winkler et al. [2018], the different levels of information to be specified include the order of feet in contact, order and times when each foot is in contact and order, times and position of each foot in contact.

- **Contact Schedule Optimization:** Griffin et al. [2019] developed a footstep planner that utilizes a planar region representation of the environment which enables footstep planning over rough terrain. When feet are modeled as discrete variables, Integer Programming can be used to optimize the contact-schedule and footholds, independent from the dynamics of the system. Ren et al. [2020] presented two optimization formulations for legged systems using Direct Collocation method and iterative Linear Quadratic Regulator (iLQR) method. The Direct Collocation-based approach generates a whole-body motion plan in a hierarchical two-phase manner with the input of state estimation and global elevation mapping yet without prior knowledge on footholds and step timings. Alternatively, a discrete-time, constrained iLQR algorithm works out an executable control policy besides a nominal optimized trajectory, however, a gait pattern should be pre-specified.

We have adopted the trajectory optimization formulation presented in Winkler et al. [2018]. The initial and desired final state of the system, the total duration T and the amount of steps $n_{s,i}$ per foot i is provided. With this information the algorithm finds a trajectory for the linear CoM position $r(t)$, its orientation $\theta(t)$, the feet motion $p_i(t)$ and the contact force $f_i(t)$ for each foot, while automatically discovering an appropriate gait pattern defined by $\Delta T_{i,j}$. Each foot's workspace is estimated by a cube of edge length $2b$, assuming that the joint constraints are not violated if every foot is inside this cube.

Each phase (swing or stance) is represented by either a constant value or a sequence of cubic polynomials with continuous derivatives at the junctions. Solely by changing the phase duration $\Delta T_{i,j}$, a completely different gait can be generated. Performing this for all feet allows to generate arbitrary gait patterns, while still using continuous decision variables. Since the phase durations are continuous, these can be readily optimized by NLP solvers and Integer Programming can be avoided. The detailed feet motion and forces parameterization, terrain height constraints and stance force constraints have been discussed in Winkler et al. [2018], hence these details are omitted from here.

2 Mechatronics

2.1 Design Considerations

To solve the problem of mobility, we looked at the various methods of manoeuvring used by hexapods and robots in general. From this, the most popular choices were to either use wheels or leg-like structures for mobility. Each method has its advantages and disadvantages, which are listed in brief below:

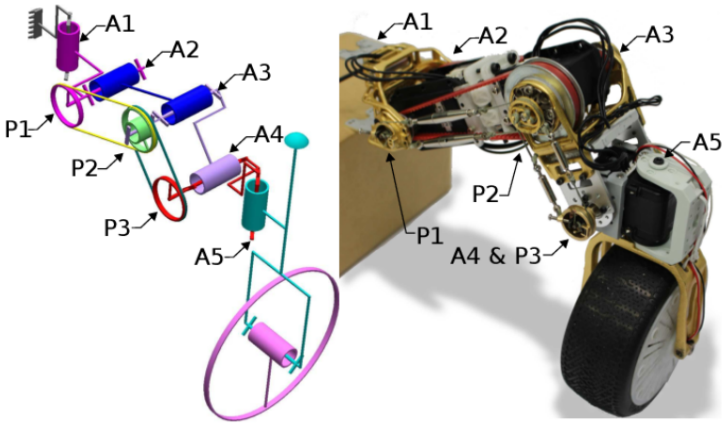
- **Legged Robots:**
  - Advantages:
    - Adaptable to various terrains
    - Can step over obstacles
    - Multiple degrees of freedom at each leg
    - Great stability while standing and moving
    - Legged hexapods mimic the movement of insects
  - Disadvantages:
    - Complex design and control
- **Wheeled Robots:**
  - Advantages:
    - Low cost and simplicity
    - Easy to steer and control
  - Disadvantages:
    - Not suitable for rough or non-flat terrains
    - Slipping due to low friction coefficient

Analysing this, we decided that we wanted the best of both worlds. Thus we decided to finalise a wheel-legged method of mobility for our robot. In an ideal wheel-legged robot, the default responsibility of movement lies on its wheels. This helps the robot move faster. Once it senses an uneven surface or a surface that is difficult for the wheels to traverse, the legs take control. As complex as it sounds, Wheel-legged robots are challenging to design and build since it has to incorporate the coordination of wheels, coordination of legs and coordination of wheels and legs.

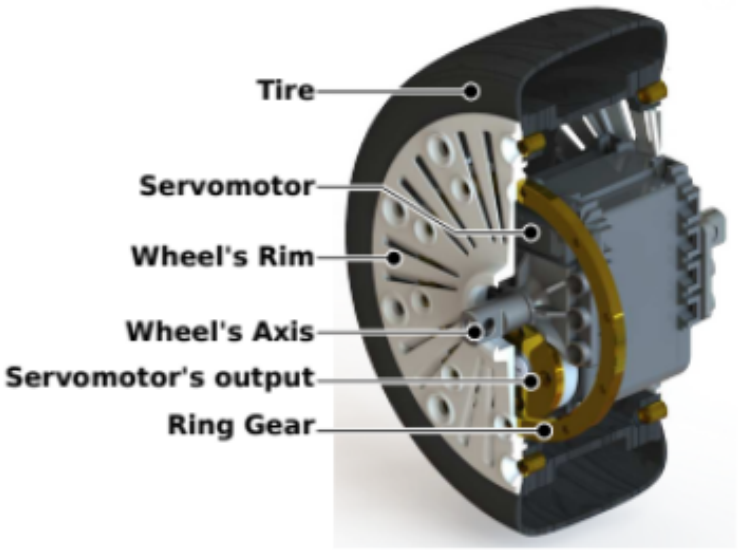
2.2 Final Design

Because the robot had to endure spiky, undulated and unpredictable terrain, we decided to go with the spider robots for high precision and degrees of control. While being able to move comfortably on these terrains, the robot should also be efficient in easier conditions as that of a flat road for which a spider robot would not be the first choice due to relatively slow waking speeds compared to wheeled robots. The combination of a spider and a wheeled robot (wheels at the foot of each leg controlled with a servo or a DC motor) seemed to suffice the above-stated needs. The core mechanisms involved in the design are described below:

- **Hip, Elbow and Ankle Joints:** Controlled by using servo motors (Position control).
- **Toothed Belt Reduction System:** t is used for inverting the position of the two servo motors (This solution embeds the two servo motors and the reduction system in a single and compact leg segment, thus reducing the inertia of the terminal segment). This mechanism also helps in maintaining the wheels perpendicular to the body of the robot.

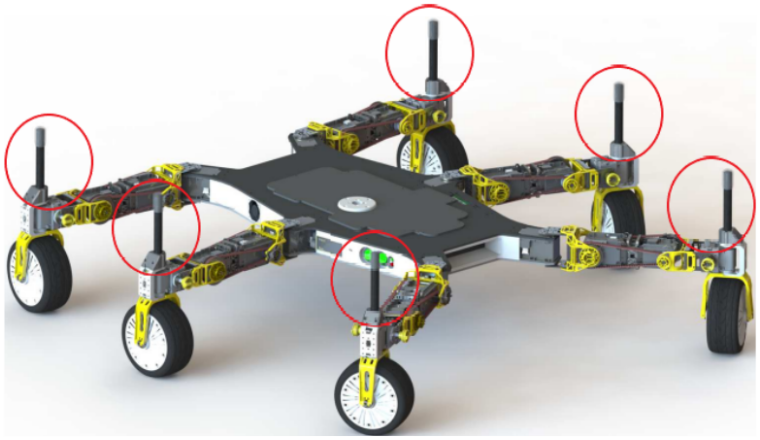


- **Braking Mechanism (embedded in wheels):** The wheel hub embeds a servo motor (or a DC motor) that actuates the wheel. Servo motors can be used by rewriting the firmware of the motor to implement a PID speed control instead of the original position control. This solution takes advantage of the servomotor compactness to undertake a compact and inertia-centred wheel design. Gears reduce the servomotor output to increase its torque, and the final ring gear is fixed to the wheel's rim that makes the wheel rolling.



- **Gear Train:** For attaining the required torque for carrying the payload. The gear trains have been chosen to reduce the motor weight and size to maintain the robot's compactness.

- **Wheel and foot switching mechanism:** If the robot flips over, the cable/belt system will prevent the robot from flipping the wheel segment. To cope with this issue, we append a foot to the top of each leg so that the robot can continue its mission by walking on its back. Thanks to the cable system, the feet will still be perpendicular to the body. The position of the tip of the foot (the contact point) is symmetrical to the wheel.



Expected Payload	60 kg
Single-Leg Payload	$\sim 16kg$
Required Torque of Servo Motor (Hip and Knee)	$\sim 18Nm$

Looking at the commercially available servo motors for the above torque requirements, we concluded that a servo motor combined with a gear set would be required to meet the torque requirements. This is because standalone motors that meet the requirements are too heavy and oversized. This would add to the weight and thus the torque requirements, resulting in a vicious cycle.

3 Computer Vision

3.1 Target Detection

Deep Neural Networks (DNNs) are popular for developing object detection algorithms. Object detection algorithms extract important information to solve computer vision problems such as object classification, localization, and recognition. In the last two decades, many deep neural network models for object detection have evolved, improving the intelligence of machine vision systems. Examples of such are faster R-CNN, Retina-Net, Single Shot MultiBox Detector (SSD) , and You Only Look Once (YOLO). Among the aforementioned state-of-the-art object detection algorithms, YOLO stands out regarding its ability to run on low-power devices, its real-time performance, and its accuracy.

- **YOLOv3:** The YOLOv3 Redmon and Farhadi [2018] algorithm would be used to identify military targets. The Yolo series algorithm is an algorithm that could detect objects quickly . YOLOv3 can achieve high precision real-time detection, which is very suitable for our application background. Its network structure is shown in Figure 4. The resolution of the input picture in the network structure diagram is 416 × 416 × 3 (in fact, it can be any resolution.), and has four labeled classes. It uses darknet-53, which removes the full connection layer, as the backbone network. The YOLOv3 is a fully convoluted network that makes extensive use of residual network structures. As shown in Figure 4, YOLOv3 consists of DBL, resn, Up-sample, and concat. DBL stands for convolution (conv), batch normalization (BN) and leaky relu activation (Leaky reu). Resn represents the n residual units (res unit) in this residual block (res block). Zero padding means using zero to fill the edge of the image. Up-sampling represents up-sampling. The concat represents the merging tensor. DBL*n represents the n DBL. The add represents the addition operation. The following y1, y2, and y3 represent feature maps with three different dimensions.

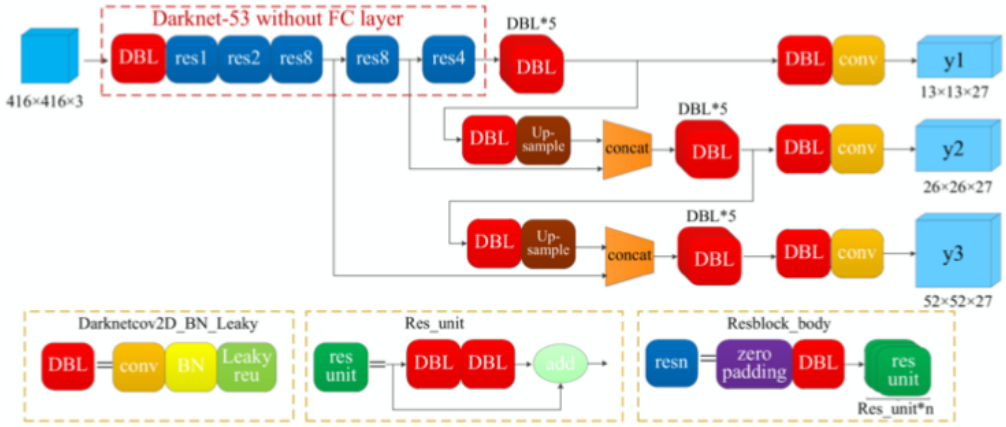


Figure 4. YOLOv3 network structure.

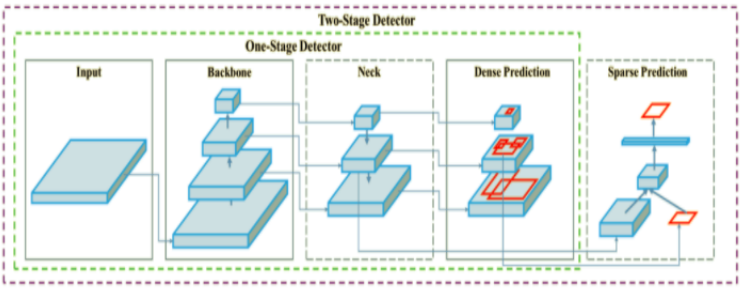
The network structure of darknet-53 is shown in Table 2. It uses successive 3 × 3 and 1 × 1 convolutional layers and some shortcut connections. The application of the shortcut connection layer allows the network to be deeper. It has 53 convolutional layers [38].

Table 2. Darknet-53 [38].

	Type	Filters	Size/Stride	Output
	Convolutional	32	3 × 3	416 × 416
	Convolutional	64	3 × 3/2	208 × 208
	Convolutional	32	1 × 1	
1×	Convolutional	64	3 × 3	
	Residual			208 × 208
	Convolutional	128	3 × 3/2	104 × 104
	Convolutional	64	1 × 1	
2×	Convolutional	128	3 × 3	
	Residual			104 × 104
	Convolutional	256	3 × 3/2	52 × 52
	Convolutional	128	1 × 1	
8×	Convolutional	256	3 × 3	
	Residual			52 × 52
	Convolutional	512	3 × 3/2	26 × 26
	Convolutional	256	1 × 1	
8×	Convolutional	512	3 × 3	
	Residual			26 × 26
	Convolutional	1024	3 × 3/2	13 × 13
	Convolutional	512	1 × 1	
4×	Convolutional	1024	3 × 3	
	Residual			13 × 13
	Avgpool		Global	
	Connected		1000	
	Softmax			

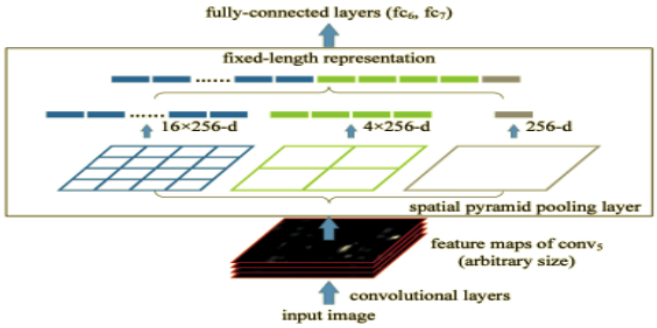
- **YOLOv4:** Yolov4 Bochkovskiy et al. [2020] is an improvement on the Yolov3 algorithm by having an improvement in the mean average precision(mAP) by as much as 10percent and the number of frames per second by 12percent. The Yolov4 architecture has 4 distinct blocks as shown in the image below, The backbone, the neck, the dense prediction, and the sparse prediction.

  The backbone is the feature extraction architecture which is the CSPDarknet53.



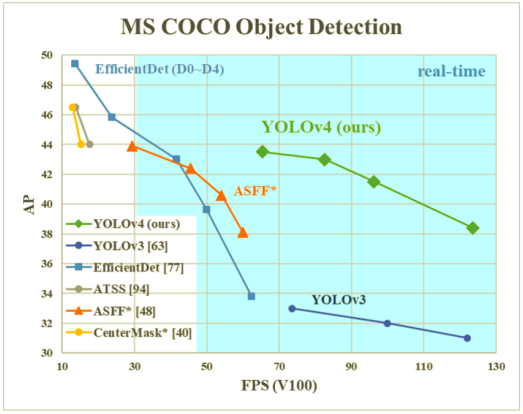
This CSPDarknet53 stands for Cross-Spatial -Partial connections, which is used to split the current layer into two parts, one to pass through convolution layers and the other that would not pass through convolutions, after which the results are aggregated.

The neck helps to add layers between the backbone and the dense prediction block(head),



which is a bit like what the ResNet architecture does. The yolov4 architecture uses a modified Path aggregation network, a modified spatial attention module, and a modified spatial pyramid pooling, which are all used to aggregate the information to improve accuracy. The image above shows spatial pyramid pooling.
The head(Dense prediction) is used for locating bounding boxes and for classification. The process is the same as the one described for Yolo v3, the bounding box coordinates(x,y, height, and width) are detected as well as the score. Remember, the main goal of the Yolo algorithm is to divide an input image into several grid cells and predict the probability that a cell contains an object using anchor boxes. The output is then a vector with the bounding box coordinates and the probabilities of the classes.

Below are the results obtained using yolov3 and yolov4 on the coco dataset for object detection, and some other detection algorithms.

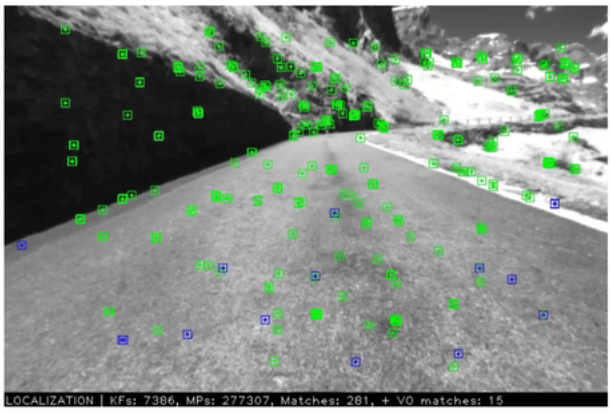


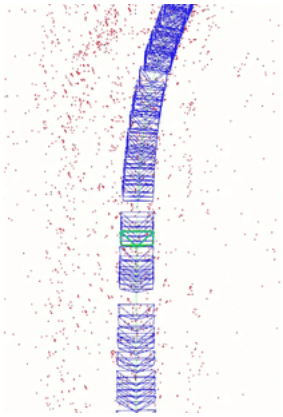
The y-axis is the absolute precision and the x-axis is the frame per second. The blue shaded part of the graph is for real-time detection(webcam, street cameras, etc), and the white is for still detection(pictures). It can be seen that the yolov4 algorithm does very well in real-time detection, achieving an average precision between 38 and 44, and frames per second between 60 and 120. The yolov3 achieves an average precision between 31 and 33 and frames per second between 71 and 120.

3.2 Simulatneous Localisation and Mapping

There are a lot of algorithms using the basic idea of Simultaneous Localization and Mapping (SLAM)Merzlyakov and Macenski [2021]. In these methods, both the localization and the map creation are made by the algorithm itself. No more information than the images themselves is used. In many cases, however, there is additional information which was taken during the capture of the images. These can be the Global Navigation Satellite System (GNSS) data about the position of the camera and the inertial measurement unit (IMU) data for the orientation of the camera. The modified algorithm got the name GPS-SLAM since the most famous system of GNSS is the Global Positioning System (GPS).

- **GPS-SLAM: An Augmentation of the ORB-SLAM Algorithm**
  Kiss-Illés et al. [2019] ORB-SLAM is a very well-functioning SLAM algorithm written by Raúl Mur-Artal et al. It uses Oriented FAST (Features from accelerated segment test) and Rotated BRIEF (Binary Robust Independent Elementary Features) feature detector (ORB) presented by Ethan Rublee et al.. We chose ORB-SLAM for this work because it is considered as "the most complete feature-based monocular visual SLAM system". With ORB-SLAM, one can create the map and the trajectory of the camera positions successfully for many datasets.

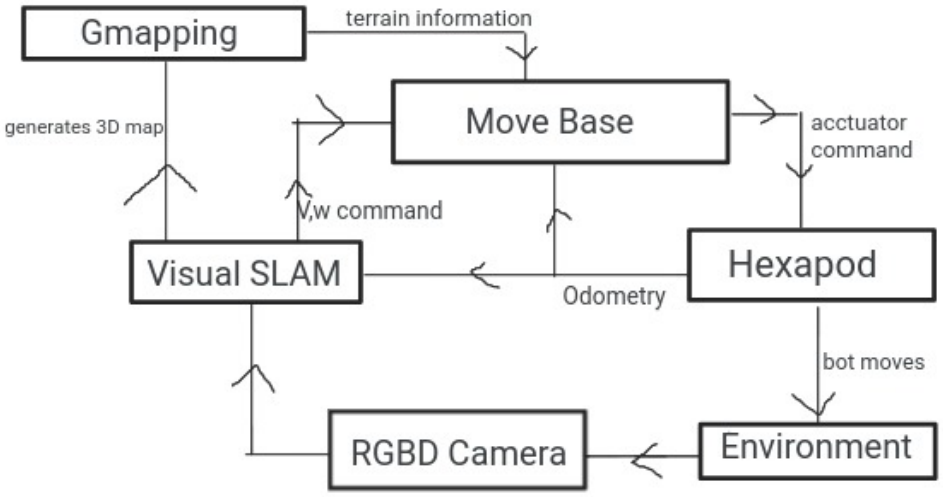




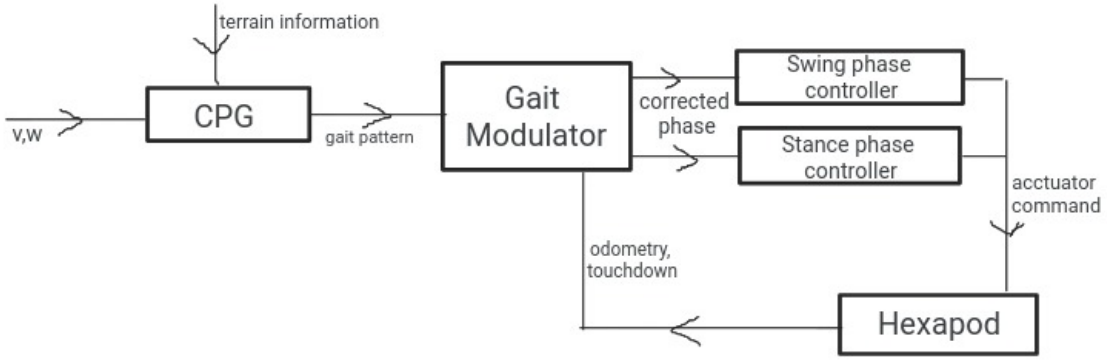
When deciding which navigation system to use in your application, it 's important to keep in mind the common challenges of robotics. Robots need to navigate different types of surfaces and routes. Our robot would need to pass through rough surfaces, climb slopes. Specific location-based data is often needed, as well as the knowledge of common obstacles within the environment.
Both visual SLAM and LiDAR can address these challenges, with LiDAR typically being faster and more accurate, but also more costly. Visual SLAM is a more cost-effective approach that can utilize significantly less expensive equipment (a camera as opposed to lasers) and has the potential to leverage a 3D map, but it 's not quite as precise and slower than LiDAR. Visual SLAM also has the advantage of seeing more of the scene than LiDAR, as it has more dimensions viewable with its sensor.

4 Overall Architecture

Having discussed about the individual modules, here we state the overall flow which is essential for each of the modules to work in sync for autonomous functioning of the hexapod. As seen in the figure below the visual SLAM generates 3D map of the environment using the data from RGBD camera. It also generates the linear and angular velocity which the bot should follow to navigate through the terrain. These commands goes to the move base which based on control strategy generates the actuator command. The functioning of move



base is as shown below. CPG (section 1.2) generates gait pattern and the trajectory based on some optimisation or learning scheme. Having generated a gait pattern we pass it on to gait pattern modulator which acts like a close loop feedback for ensuring that phases are corrected based on the current state of the hexapod. This is very necessary for maneuvering on rough terrain. Once phases are corrected we have a defined position or force for the foot. This is used to generate actuator command using swing or stance controller for the respective phases.



These architecture are based on our understating of the control so far. It might have to be changed appropriately at the implementation phase based on the results.

References

M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, and S. Schaal. Fast, robust quadruped locomotion over challenging terrain. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2665–2670, May 2010. URL http://www-clmc.usc.edu/publications/K/kalakrishnan-ICRA2010.pdf. clmc.

Alexander Winkler, Ioannis Havoutis, Stephane Bazeille, Jesus Ortiz, Michele Focchi, Rüdiger Dillmann, Darwin Caldwell, and Claudio Semini. Path planning with force-based foothold adaptation and virtual model control for torque controlled quadruped robots. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6476–6482, 2014. doi: 10.1109/ICRA.2014.6907815.

Diego Pardo, Michael Neunert, Alexander Winkler, Ruben Grandia, and Jonas Buchli. Hybrid direct collocation and control in the constraint-consistent subspace for dynamic legged robot locomotion. 07 2017. doi: 10.15607/RSS.2017.XIII.042.

Alexander W. Winkler, C. Dario Bellicoso, Marco Hutter, and Jonas Buchli. Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robotics and Automation Letters*, 3:1560–1567, 2018.

Robert J. Griffin, Georg Wiedebach, Stephen McCrory, Sylvain Bertrand, Inho Lee, and Jerry Pratt. Footstep planning for autonomous walking over rough terrain. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, page 9–16. IEEE Press, 2019. doi: 10.1109/Humanoids43949.2019.9035046. URL https://doi.org/10.1109/Humanoids43949.2019.9035046.

Jiming Ren, Shuo Yang, Zhaoyuan Gu, and Howie Choset. Trajectory optimization for the legged system, 09 2020.

Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

Alexey Merzlyakov and Steve Macenski. A comparison of modern general-purpose visual slam approaches, 2021.

Dániel Kiss-Illés, C. Barrado, and E. Salamí. Gps-slam: An augmentation of the orb-slam algorithm. *Sensors (Basel, Switzerland)*, 19, 2019.