1. Описание на алгоритми, апаратна или програмна част
   1. ПИД

Пропорционално-интегрално-диференциалният (ПИД) регулатор е регулатор с три съставки, който се използва от дълго време в областта на автоматичното управление (от началото на 20-ти век). Заради интуитивността си и относителната си простота, той се е превърнал в стандартния регулатор за индустриалните приложения. Освен задоволителната си работа, ПИД регулаторът осигурява широк обхват от процеси. Регулаторът се е развивал заедно с развитието на технологиите и днешните му реализации много често са в цифрова форма вместо реализациите с пневматични или електрически компоненти. Той може да бъде открит на практика при всички типове управляващо оборудване или като самостоятелен регулатор или като функционален блок в ПЛК и в разпределените системи за управление. Успехът на ПИД регулаторите се дължи също и на факта, че те често са един фундаментален компонент от по-усъвършенстваните управляващи системи, който може да бъде приложен, когато основният закон за управление не е достатъчен за постигане на изискваните експлоатационни качества или трябва да бъде решена по-сложна задача за управление.

ПИД контролера изчислява грешката между измерената и желаната стойност на управляващият сигнал. Целта на контролерът е да минимизира възможно най-бързо тази грешка и да достигне установен режим или в случаят стабилизиран полет. По специфично ПИД контролерът взима данните измерени от сензорите на АПМ 2.5(жироскоп, акселерометър, компас, барометър и т.н.) и ги сравнява с очакваните или желани стойности. Изходният сигнал се подава на ШИМ изходите на процесора. ШИМ сигналът се преобразува от контролерите на скоростта (ESC) в трифазен променлив сигнал и се изпраща към моторите.

A proportional-integral-derivative controller (PID controller) is a generic control loop feedback mechanism (controller) widely used in industrial control systems. A PID controller calculates an "error" value as the difference between a measured process variable and a desired setpoint. The controller attempts to minimize the error by adjusting the process control inputs.  
  
In specific multicopter terms this means the PID software will be taking data measured by the sensors on the flight controller (gyros / accelerometers etc) and comparing that against expected/desired values to alter the speed of the motors to compensate for any differences and maintain control. 

Depending on your flight controller there will be PID's associated with a number of flight modes.   
You should always start tuning standard PID's first,  Rate Roll, Rate Pitch and Rate Yaw. Until you get these stable then there little point in moving on to the others.

The ArduCopter already includes this kind of autopilot, but it is a PI, Proportional-Integral, controller that stabilizes the angular rates.

The control loop of the tricopter can be seen as one inner and one outer loop, see Figure 2.4. The inner loop is a faster one and controls the rotational rates of the tricopter. The frequency of this loop is 50Hz, which gives a hard deadline of 20 ms the loop has to compute the input signal to the system. The outer loop is a slower one and this controls translational position, translational velocity and rotational angles of the tricopter. In this thesis, only the inner loop is considered.

It was very interesting to see that even those very simple controllers are able to stabilize and even provide robust performance when a suitable architecture is chosen.

What we are trying to do with this controller is remove the need for the different stages of altitude changing that was in there before. This controller can handle both the altitude hold function as well as the fast altitude changes. We have tested it up to 5 m/s. This lets us do the very fast reduction in altitude I think you were asking for. It also does this with very little overshoot.

We use the square root of distance relationship to slow the airframe at reasonable accelerations to remove the very large accelerations that a linear distance vs velocity relationship applied. We took this one step further by combining the linear relation P loop with the square root relationship. This provides a smooth transition between the two modes in both acceleration and velocity.

The linear distance calculation is the distance that the square root relationship needs to be moved in order for the linear velocity relationship to be a tangent to the square root velocity relationship. This provides a step free transition between the two at the linear distance x 2. Acceleration of the two curves is also equal at this distance meaning that the motors don't pulse as we swap between the two curves.

 The 250 cm/s/s is the maximum acceleration that will be required of the copter while still allowing the copter to stop without overshoot. The copter will not reach these accelerations unless it is instructed to travel over approximately 125 cm/s. This shouldn't happen during Alt Hold but can happen during Altitude changes. This limit was chosen because it should be achievable for all reasonably designed copters. To not be capable of this acceleration the copter would have a cruise throttle of over 700.

Using this controller you can still move the requested altitude slowly however if we need to stop or change direction it is able to still follow that request. In fact, the controller is not limited in the acceleration it can ask for. In an emergency this controller will be able to do it in less than 1 second and 2.5m. (I realize these are simple calculations and this situation only occurs if the user asks for stupid things).

We now have different speed limits for up and down but currently they are set to the same thing. We did this because the controller can now handle much faster rates so we didn't think we needed to limit it to such small values going down. I assume this was done for safety.

The final thing I should mention is the design of the controller with respect to the pid loops. This is a (P/sqrt) -> PD -> PI (Position -> Velocity -> Acceleration) controller by default. However, there may be advantages to using (P/sqrt) -> PD -> PID with correct filtering.

 So why no I term in position. This is because the I terms in these loops will reduce the following error when using this controller with a moving set point. The current design of the controller ensures that the distance between the moving altitude request and the current altitude is enough for the controller to stop the aircraft in that distance without accelerating beyond 250 cm/s/s. This is how we achieve such a small overshoot from very fast decent rates. This is also why we don't use an I term in the Velocity controller, we don't want overshoots due to I term build up.

The other parameter that is critical in this function is the velocity D term. This has the effect of minimizing the error between the requested velocity and the desired velocity during deceleration. This is set as large as it can be without noticing any jerkiness or oscillation during altitude changes.

Each component of the controller does something different. The Proportional control is assigned directly to the error variable we're trying to control (in this case, the yaw rate, or the speed that the helicopter is turning at), and directly assigns a response based on the error in the system. The intergal component integrates the error (adds it up over time) to force the error to zero when the proportional controller can't correct every error perfectly. The derivative component monitors how fast the error is changing, and applies feedback to slow that change down to stop oscillations, or dampen the system.

Basically, the proportional component directly assigns a response based on how large the system error is (desired rate - actual rate). For yaw, the reason the heli spins is because a proportional controller can't adjust its self for non-linearities in how a helicopter flies. If the response isn't modeled perfectly (which it can't be because helis are so non-linear), there will be a constant rate error in the output. Basically, it will always slowly spin.

The integral component is what's designed to fix that. Just like a mathematical integral, the error integral adds up over and over every cycle. The response of the integral controller gets stronger and stronger the longer the error sticks around. For nonlinear systems, this positive feedback makes up for imperfections in the proportional control, and forces the error to zero.

The drawback of integral control is that while that error is building up, once the system hits zero error, that integral component is still built up, and applying its control to the system. As a result, the integral controller causes the system to fly through the target desired value and overshoot. This causes oscillations in the system, necessitating a third controller component to counter those unwanted effects.

That's what the D component is for. The D component just looks at how fast the error is changing, and applies negative feedback against that change. The end result is when the error is approaching zero very quickly as a result of the I component doing its job properly, the D component subtracts against the error that's driving the other controllers, giving them a smaller value to work with. This makes the other controllers slow their roll as they approach the disired value, eliminating the oscillations. One example of a mechanical derivative controller is the shock absorbers in your car's suspension.

**What PIDs mean for performance**

Stabilize is used for all modes except Acro. It's intended to be symmetrical, but pitch and roll have been broken out so you can fine tune a copter that is bearing a load in 1 axis. This might be a result of using two battery packs hanging on either side of the frame, or a very long thin pack. If one axis has a higher moment of inertia than the other axis, your ideal tuning parameters will not be symetrical.

* STABILIZE\_P: The rotation rate at which you want to correct any errors. The higher this is faster the copter will attempt to achieve the desired attitude.
* STABILIZE\_I: Used to account for CG variations, weak motors or persistent external forces.
* STABILIZE\_Imax: Maximum amount the copter can compensate for these imbalanced forces.
* RATE\_P: The most important value! This gain controls how much thrust you need to output to achieve the desired rate of rotation. High thrust/weight copters will require a lower value, lower thrust/weight will require a higher value. A too-high value will oscillate around 5-10hz. You want this value to be slightly lower than the value that causes the oscillation. Aggressive tuning may result in exactly 1 overshoot and return, which is considered acceptable. Use CH6 tuning to adjust in the air for best performance.
* RATE\_I: Used to help a copter achieve a desired roll rate. Not used by default as this can be very difficult to tune properly and can be confusing. If you are just starting out, set this term to 0.
* RATE\_IMAX: The maximum amount of Rate\_I that can build up. This is also not used in a basic setup. Having a 0 iMax will make Rate\_I completely ineffective, no matter how high the Rate\_I is.

Yaw is used to hold a particular Yaw angle. If your copter wants to spin naturally, you won't be able to hold an exact heading. You will instead drift a few degrees until P gets significantly high to stop rotation.

* STABILIZE\_YAW\_P: The desired rate at which the copter will return to the target heading. If this is too high, it could cause an oscillation.
* STABILIZE\_YAW\_I: Acts like a trim to overcome poor copter balance. Defines time it takes to achieve max value. Higher = faster.
* RATE\_YAW\_P: Used to the amount of control authority the AC2 can use to achieve zero yaw rate. If this is too low, you will never be able to stop a rotation. If this is too high, it will yaw-oscillate.
* RATE\_YAW\_I: Not used

LOITER is used to control how much the copter will pitch towards the Loiter target while trying to hold a position.

* LOITER\_P: The rate at which the copter will move towards the target point. If this is not high enough, the copter will not be able to fight high winds and will drift. If it's too high, it will oscillate around the target.
* LOITER\_I: This will help the copter fight winds while having a zero error. However use it with caution because it will also cause an oscillation if it's too high.
* LOITER\_IMAX: The maximum possible buildup of Loiter\_I

NAV\_WP is used to control the rate of speed of the copter towards the target.

* NAV\_WP\_P: we use our speed (4m/s as defined by WP\_SPEED\_MAX) offset as the error. high numbers = more pitch to achieve the desired speed
* NAV\_WP\_I: Allows us to ramp up against the wind. Higher value ramps faster.
* NAV\_WP\_IMAX: Amount of Pitch we can add to overcome wind

Alt Hold is used to hold a position using the relatively noisy Barometric pressure sensor.

* Altitude hold P: Used to convert altitude error in centimeters to a desired climb\_rate in centimeter/second. Higher = faster climb rate.
* Altitude hold I: Used to account for a copter having trouble holding altitude, usually due to a low voltage battery.
* Altitude hold IMAX: Amount of throttle we can adjust (units: 1000 = 100%)

Throttle Rate is used to dampen the copter and control the rate of climb

* Throttle Rate P: amount of throttle output used to change the climb rate
* THROTTLE\_I: compensates for error in achieving desired climb rate (zero by default. We use Altitude hold I to do most of the work.)
* THROTTLE\_IMAX: Amount of Throttle\_I we can add or remove to achieve desired climb rate.

Описание на параметрите на системата

**Rate Roll**

* **P** - Too much rate P will oscillate quickly, and cause to copter to sound angry under stick input, bouncing rather than smoothly following your inputs. It will also shake more at full throttle and under hard turning. Not enough you will not feel like you have full control,  it will feel lazy and be very easy to over correct with your inputs, inputs will feel delayed.
* **I** -  Too much rate\_I will oscillate if you get high enough (a much slower oscillation than a rate\_P shake).  But quite a long while before it oscillates it will have other detrimental effects on flight performance, like a sluggish feeling or a tendency to flip over on take-off. Not enough will cause the copter to get pushed by a constant wind, then it will fight back using just P.  It will not hold a very firm angle during forward flight and will need more correction. This will not be as smooth as it could be in either case.
* **D** - Too much rate\_D will cause fast oscillations, you will see a twitch forming then a fast buzzing oscillations. Not enough rate\_D will result in you not being able to dial enough rate\_P in, you will then suffer the effects of having rate\_P too low
* **IMAX** - The maximum possible build up of Roll

**Rate Pitch**

* **P** - Too much rate P will oscillate quickly, and cause to copter to sound angry under stick input, bouncing rather than smoothly following your inputs. It will also shake more at full throttle and under hard turning. Not enough you will not feel like you have full control,  it will feel lazy and be very easy to over correct with your inputs, inputs will feel delayed.
* **I**- Too much rate\_I will oscillate if you get high enough (a much slower oscillation than a rate\_P shake).  But quite a long while before it oscillates it will have other detrimental effects on flight performance, like a sluggish feeling or a tendency to flip over on take-off. Not enough will cause the copter to get pushed by a constant wind, then it will fight back using just P.  It will not hold a very firm angle during forward flight and will need more correction. This will not be as smooth as it could be in either case.
* **D** - Too much rate\_D will cause fast oscillations, you will see a twitch forming then a fast buzzing oscillations. Not enough rate\_D will result in you not being able to dial enough rate\_P in, you will then suffer the effects of having rate\_P too low
* **IMAX** - The maximum possible build up of Pitch

**Rate Yaw**

* **P** - Should be set higher get more aggressive control and lower to slow reaction time.
* **I** -
* **D** -
* **IMAX** - The maximum possible build up of Yaw

**Stablize Roll**

* **P** - The desired rate at which the copter will return to level from roll.  If this is too high, it could cause an oscillation.
* **I** - Acts like a trim to overcome poor copter balance. Defines time it takes to achieve max value. Higher = faster.
* **IMAX** -

**Stabilize Pitch**

* **P** - The desired rate at which the copter will return to level from pitch forward or back.  If this is too high, it could cause an oscillation.
* **I** - Acts like a trim to overcome poor copter balance. Defines time it takes to achieve max value. Higher = faster.
* **IMAX** -

**Stabilize Yaw**

* **P** - The desired rate at which the copter will return to the target heading.  If this is too high, it could cause an oscillation.
* **I** - Acts like a trim to overcome poor copter balance. Defines time it takes to achieve max value. Higher = faster.
* **IMAX** -

**Loiter Speed**

* **P** - Too low and Multi will overshoot position.
* **I**- Overcomes wind to hold position (turn to zero in no wind conditions to tune P)
* **IMAX** - Maximum speed to position.

**Rate Loiter**

* **P** - The rate at which the copter will move towards the target point.  If this is not high enough, the copter will not be able to fight high winds and will drift.  If it's too high, it will oscillate around the target.
* **I**- This will help the copter fight winds while having a zero error.  However use it with caution because it will also cause an oscillation if it's too high.
* **D**
* **IMAX** - The maximum possible build up of Loiter

**Throttle Rate**

* **P** - amount of throttle output used to change the climb rate
* **I** - compensates for error in achieving desired climb rate (zero by default.)
* **D**
* **IMAX**  - The maximum possible build up of throttle.

**Altitude Hold**  
   
If you are having problems tuning ALT HOLD, make sure you have minimised all vibrations of the frame and mounted the FC on foam or gel etc. Also try setting AHRS\_MPU6K\_FILTER to 20 in advanced params in Mission Planner

* **P** - Used to convert altitude error in centimeters to a desired climb\_rate in centimeter/second. Higher = faster climb rate
* **I** - Used to account for a copter having trouble holding altitude, usually due to a low voltage battery.
* **IMAX** - Amount of throttle we can adjust  (units: 1000 = 100%)

Ардукоптер Контрол обяснение

Има много различни методики за управление на летателни многороторни апарати и най-вече на три- и четирикоптери обекти. В литературата се набляга основно на пропорционално, интегрално диференциалният и линееен квадратичените регулатори за управление. Направени са опити и за интегриране на предказващ регулатор(MPC)[5] в АПМ2.5.

(Bouabdallah et al. 2005) have used this controller to stabilize the attitude of the quadrotor around the hover position. The controller was designed using linearized model of the quadrotor in the hover trim point. The controller was developed using the nonlinear Simulink model and it was verified on the physical system. The resulting controller was able to stabilize the physical system within three seconds.The linearity of the controller constraints its use only around the hover trim point. Strong perturbation from this positions leads to loss of control. (Hoffmann et al. 2007) have used PID control for controlling attitude, altitude and position. Results were satisfactory, but the quadrotor has not performed any aggressive maneuvers and the disturbance rejection of the control system was not very good.

* 1. Линейно квадратичен регулатор

(Castillo et al. 2005) have implemented this kind of controller. During simulation the controller has performed satisfactory. When strong perturbation was introduced the controller due to its linearity was not able to stabilize the system. On the physical model, this controller was not able to stabilize the system at all. (Bouabdallah et al. 2005) have implemented LQR controller using multiple trim points. Unfortunately they have not implemented the motor dynamics into the model. This lead to worse performance than their already mentioned PID controller.

* 1. Предсказващ регулатор (MPC)